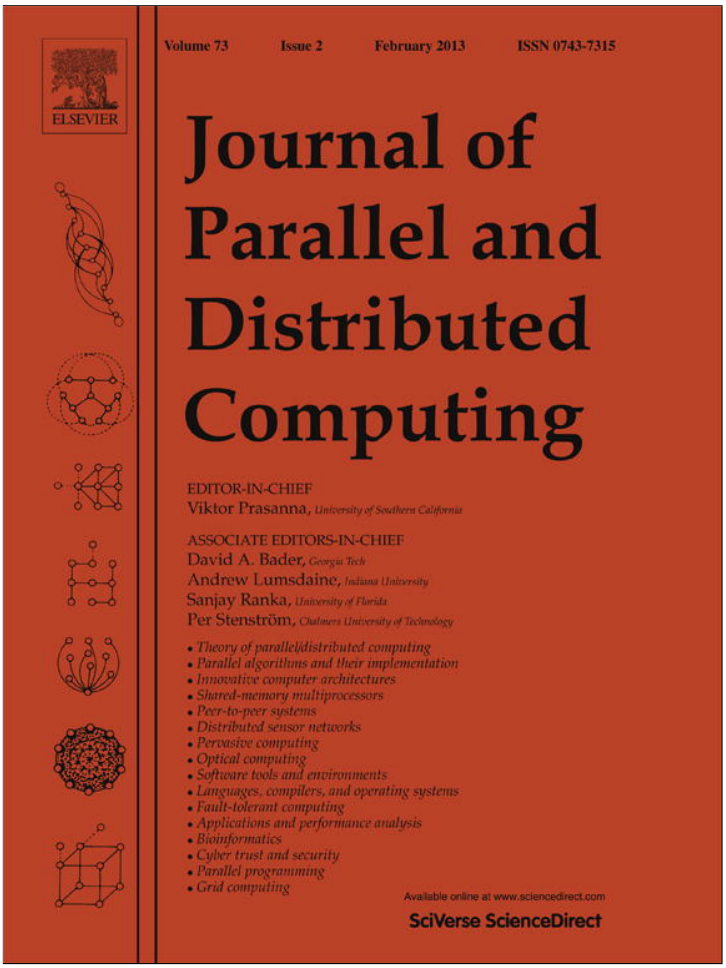

J. Parallel Distrib. Comput. 73 (2013) 256–272

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Comparisons of air traffic control implementations on an associative processor with a MIMD and consequences for parallel computing

Man (Mike) Yuan [a,*], Johnnie W. Baker [a], Will C. Meilander [b]

[a] Department of Computer Science, Kent State University, Kent, OH 44242, United States

[b] Kent State University, Gainesville, GA, United States

ARTICLE INFO

Article history:
Received 20 November 2011
Received in revised form
10 May 2012
Accepted 27 May 2012
Available online 27 July 2012

Keywords:
Air traffic control (ATC)
SIMD
MIMD
Instruction stream (IS)
Real-time systems
Associative processor (AP)
Conflict detection and resolution (CD&R)
ClearSpeed
Multicore processor
OpenMP
Predictability
Worst case execution time
Federal Aviation Administration (FAA)
Multiprocessor
NP-complete
Deterministic hardware
STARAN
ASPRO
Associative SIMD
Static scheduling

ABSTRACT

This paper has two complementary focuses. The first is the system design and algorithmic development for air traffic control (ATC) using an associative SIMD processor (AP). The second is the comparison of this implementation with a multiprocessor implementation and the implications of these comparisons. This paper demonstrates how one application, ATC, can more easily, more simply, and more efficiently be implemented on an AP than is generally possible on other types of traditional hardware. The AP implementation of ATC will take advantage of its deterministic hardware to use static scheduling. The software will be dramatically smaller and cheaper to create and maintain. Likewise, a large AP system will be considerably simpler and cheaper than the MIMD hardware currently used. While APs were used for ATC-type applications earlier, these are no longer available. We use a ClearSpeed CSX600 accelerator to emulate the AP solutions of ATC on an ATC prototype consisting of eight data-intensive ATC real-time tasks. Its performance is compared with an 8-core multiprocessor (MP) using OpenMP. Our extensive experiments show that the AP implementation meets all deadlines while the MP will regularly miss a large number of deadlines. The AP code will be similar in size to sequential code for the same tasks and will avoid all of the additional support software needed with an MP to handle dynamic scheduling, load balancing, shared resource management, race conditions, false sharing, etc. At this point, essentially only MIMD systems are built. Many of the advantages of using an AP to solve an ATC problem would carry over to other applications. AP solutions for a wide variety of applications will be cited in this paper. Applications that involve a high degree of data parallelism such as database management, text processing, image processing, graph processing, bioinformatics, weather modeling, managing UAS (Unmanned Aircraft Systems or drones) etc., are good candidates for AP solutions. This raises the issue of whether we should routinely consider using non-multiprocessor hardware like the AP for applications where substantially simpler software solutions will normally exist. It also raises the question of whether the use of both AP and MIMD hardware in a single hetergeneous system could provide more versatility and efficiency. Either the AP or MIMD could serve as the primary system, but could hand off jobs it could not handle efficiently to the other system.

1. Introduction

The Air Traffic Control (ATC) system is a real-time system that continuously monitors, examines, and manages space conditions for thousands of flights by processing large volumes of data that are dynamically changing due to reports by sensors, pilots, and controllers, and gives the best estimate of position, speed and heading of every aircraft in the environment at all times. The ATC software consists of multiple real-time tasks that must be completed in time to meet their individual deadlines. The Federal Aviation Administration (FAA) has put tremendous efforts on finding a predictable and reliable system to achieve *free flight* which would allow pilots to choose the best path to minimize fuel consumption and time delay rather than following pre-selected flight corridors [45,61,83]. In the past, solutions to this problem have been implemented on multicomputer systems with records for various aircraft stored in the memory accessible to the processors in this system. The dispersed nature of this dynamic ATC system and the necessity of maintaining data integrity while providing rapid access to this data by multiple instruction streams (*MIS*) increases the difficulty and complexity of handling air traffic control. It is difficult for these MIMD implementations to satisfy reasonable predictability standards that are critical for meeting the strict certification standards needed to ensure safety for critical software components. Massive efforts have been devoted to finding an efficient MIMD solution to the ATC problems since 1963.

* Corresponding author.
E-mail addresses: myuan@cs.kent.edu (M. Yuan), jbaker@cs.kent.edu (J.W. Baker), wllcm@att.net (W.C. Meilander).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.05.009

The most critical issue of *free flight* is conflict detection and resolution (CD&R). The performance of all CD&R algorithms available depends on aircraft state estimation according to the comprehensive survey of Kuchar and Yang [47]. The Kalman filter [9,20] is the central algorithm for the majority of all modern tracking systems, known as $\alpha - \beta, \alpha - \beta - \gamma$ filters. The major problem with the single Kalman filter is that it does not predict well when the aircraft makes an unanticipated change of flight mode such as making a maneuver, accelerating, etc. [40–42,52]. Many adaptive state estimation algorithms have been proposed [54,48,10,77]. The Interacting Multiple Model (IMM) algorithm [19,51] runs two or more Kalman filters that are matched to different modes of the system in parallel. It uses a weighted sum of the estimates from the bank of Kalman filters to compute the state estimate. IMM and its variants have been applied to single and multiple aircraft tracking problems in [54]. However, it becomes inaccurate for tracking multiple aircraft as the number of aircraft increases. Current MIMD implementations of this algorithm are computationally very intensive. Hwang et al. [41] proposed that the flight mode likelihood function can be used to improve the estimation results of the IMM algorithm. The likelihood function uses the mean of the residual produced by each Kalman filter. A heuristic algorithm in [52] that evaluates correlation error values has been shown to provide better results than the Kalman filter [52].

A comprehensive survey of the CD&R algorithms is presented in [47]. Menon et al. [60] formulate conflict resolution as a multi-participant optimal control problem. Using parameter optimization and state constrained trajectory optimization, they compute a conflict resolution trajectory for two different cost functions: deviation from the original trajectory and a linear combination of total flight time and fuel consumption. Their method results in 3*D* optimal multiple-aircraft conflict resolution. In general, the optimization process is computationally intensive and is difficult to implement in real-time. In [46], Krozel et al. propose one centralized strategy that is controller-oriented and two decentralized strategies that are user-oriented. In the centralized approach, a central agent analyzes the trajectories of the aircraft and determines resolutions. In the two decentralized strategies, each aircraft resolves its own conflicts as they are detected. In [83], Yang and Kuchar propose a conflict alerting logic based on sensor and trajectory uncertainties, with conflict probability based on Monte Carlo simulation. Chiang et al. [25] propose CD&R algorithms from the perspective of computational geometry. Paielli and Erzberger [63] and Prandini et al. [69] propose analytic algorithms for computing probability of conflict. Many of the algorithms consider only two aircraft. For example, Krozel et al. [46] show that neither their centralized nor decentralized CD&R algorithms can guarantee safety for multiple aircraft when the number of aircraft is growing. Furthermore, many algorithms propose optimization schemes that are not guaranteed to be completed within real-time deadlines. Due to the increasing number of FAA problems, FAA is inviting proposals for new and efficient CD&R [32]. A more extensive literature review and detailed background information can be found in Chapters 2–3 of the author's dissertation [84].

Instead of using the traditional MIMD approach, we implement a prototype of the ATC system on an enhanced SIMD hardware system called an associative processor (AP) or associative SIMD [11,59,67,72], where interactions are much simpler and more efficiently controlled, due in large part to avoiding the need to coordinate the interactions of multiple instruction streams. We have chosen eight key ATC tasks to use in our ATC prototype, namely report correlation and tracking, cockpit display, controller display update, sporadic requests, automatic voice advisory, terrain avoidance, conflict detection and resolution (CD&R) and final approach (runway optimization). The selection of the specific tasks was based on information in the following references: FAA Grants for Aviation Research Program Solicitation [33], FAA's NextGen Implementation Plan [34], and the FAA 1963 ATC Specifications [35]. Ref. [35] indicates that the tasks we selected were similar to the ones selected in 1963 for implementation by industries interested in competing for the job of implementing ATC for FAA. Ref. [33] describes the FAA's efforts to improve the aircraft capacity of the airspace while maintaining high safety standards and aircraft safety technology for conflict detection and resolution. These indicate that the tasks of the type that we have selected are key to ATC implementation. Further, the NextGen plan [34] discusses the new standards for ATC including developing capabilities in traffic flow management, dynamic airspace configuration, separation assurance, super density operations, and airport surface operations. The purpose is to achieve a safe, efficient and high-capacity airspace system. The following subtopics in [34] address current research that needs improvement: comprehensive analysis of uncertainties in the National Airspace System; air traffic management functional allocation using advanced computing and networking; guarantee safety of air-to-air and air-to-ground. The type of activities discussed there will require the use of tasks similar to the ones that we have selected for implementation, which show that our eight ATC tasks have already captured most of the workload of ATC, and nothing major is missing that may impact the ATC performance. In our careful search of literature for publications on ATC, the ones that kept re-occurring in the ATC literature we found were all included in the tasks we chose for our prototype to use for benchmarking.

Several of our previous papers [59,57,58,85] have used the AP to manage ATC computation for an ATC sector. Similar SIMD parallel approaches have been used for collision avoidance algorithms between multiple agents for real-time simulations in [38]. We used ClearSpeed CSX600 to emulate an AP in our previous work [86]. However, the assumed maximum number of aircraft being tracked is 4000 IFR (instrument flight rules) aircraft and 10 000 VFR (visual flight rules) aircraft, for a total of 14 000 aircraft [86]. Because the emulation tool CSX600 can only process a small number of aircraft, an ideal AP system would have to be a lot larger than the CSX600.

Our paper [86] has implemented two ATC tasks, i.e., report correlation tracking and conflict detection and resolution (CD&R) on CSX600, and compared the performance of only one task, report correlation and tracking, on both CSX600 and MIMD. However, in the comparison tests, only one task was executed repeatedly, without any competition from other tasks. Also, no adjustment was included for the greater combined computational speed of MIMD. In this paper, we not only implement the 8 tasks in CSX600, but also schedule three tasks (report correlation and tracking, terrain avoidance and CD&R) on both CSX600 and an 8-core MIMD with the fastest host-only version implemented using OpenMP [62]. Our results show that some deadlines are regularly missed by these tasks when implemented on a MIMD, while the AP implementation meets all deadlines for the tasks that can be statically scheduled. The results indicate that the AP emulation has much better scalability, efficiency and predictability than the MIMD implementation. Moreover, the proposed AP solution will support accurate predictions of worst case execution times and will guarantee that all deadlines are met. The author's dissertation [84] has a more complete explanation of the consequences and implications for parallel computing in more detail.

There are three basic types of parallel systems that have been called SIMD. The first is the traditional SIMD and includes the Goodyear Aerospace MPP, Thinking Machines CM-2, and the MasPar. These parallel computers perform the same operation on multiple data values simultaneously. The second type is called vector machines or vector processing machines. They involve the

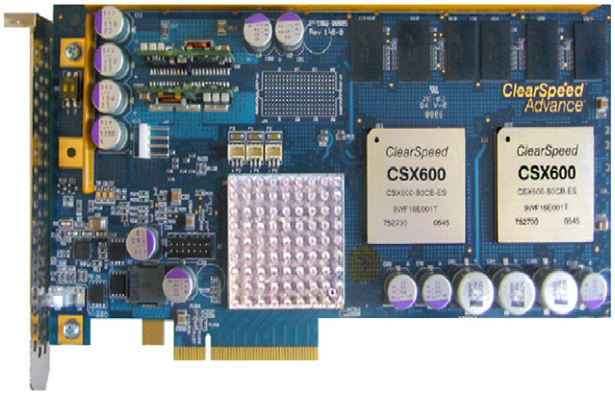

Fig. 1. CSX600 accelerator board.

use of pipelined processors and include the large CRAY vector machines. The third and most recent SIMD type are systems that are sometime collectively called short vector machines. They evolved from desktop computers as they became more powerful and capable of supporting real-time gaming and video programming. Our focus here is strictly on the traditional SIMD type.

This paper is organized as follows. Section 2 gives an overview of the CSX600 architecture and programming concepts. We discuss emulating the AP on the CSX600 in Section 3. Section 4 discusses some issues concerning the property of AP. In Section 5, the advantages of AP over MIMD are discussed. The overall system design and static scheduling is illustrated in Section 6. Section 7 describes the algorithms for 8 key ATC real-time tasks implemented on CSX600, which can be mapped to AP. Experimental results are presented in Section 8 and observations and consequences of the preceding results are in Section 9. Finally, Section 10 concludes this paper and discusses some work planned for the future.

2. Overview of ClearSpeed CSX600

The ClearSpeed accelerator board shown in Fig. 1 is a PCIe card equipped with two CSX600 coprocessors. The CSX600 board is a multi-core processor with two CSX600 coprocessors, each with 96 processing elements (PEs) connected in the form of a one-dimensional array. At present, we are only using one of the two coprocessors in order to obtain a more SIMD-like environment. This multi-core section is called a multi-threaded array processor (MTAP) core, and the architecture is shown in Fig. 2. The programmer only has to provide a single instruction stream, and the instructions and data are dispatched to the execution units that have two parts: one is a mono unit that functions as a control unit and processes sequential instructions, and the other is a poly unit that processes parallel instructions and values and has 96 PEs. At each step, all active PEs execute the same command synchronously on their individual data. Each PE has its own local memory of 6 kB, a dual 64-bit FPU, its own ALU, integer MAC, registers and I/O. The PEs operate at a clock speed of 210 MHz. The aggregate bandwidth of all PEs is specified to be 96 GB/s, which is for on-chip memory. Since the parallel bandwidth between the PEs and their memory is (no. of PEs) × (memory-PE bandwidth of each PE), this bandwidth increases as the number of PEs increase. This provides an extremely wide bandwidth for SIMDs with a large number of PEs. This allows SIMDs to avoid the von Neumann bottleneck, since a large number of PEs can access their memory in the same step without any slowdown due to message passing or shared memory access time. Further information on the hardware architecture can be found in the documentation [26].

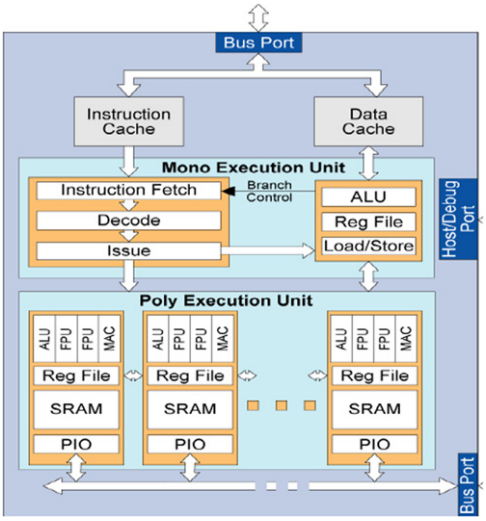

Fig. 2. MTAP architecture of CSX600.

The ClearSpeed accelerator provides the C^n language as the programming interface for the CSX600 processors. It is very similar to the standard C programming language. The main difference is that it introduces two types of variables, namely mono and poly variables. The mono variables are equivalent to common C variables and are used by the control unit. A poly value is a parallel variable, with the ith value stored in the ith processor; moreover, all these values are stored in the same memory location in their respective processor. Further information on the associated software can be found in the documentation [27,28]. Here we will use three of the library functions for data transfer on the ClearSpeed. The command *memcpym2p* copies from mono to poly memory and the command *memcpyp2m* copies from poly to mono memory. The third one, *swazzle*, moves data between adjacent PEs using the *swazzle* network, which is a ring network connecting all the processors together. More details of usages of library functions can be found in the documentation [28,37].

3. Emulating the AP on the ClearSpeed CSX600

An associative processor (AP) [59,72] is a SIMD system with several additional useful hardware enhancements that simplify supporting the real-time ATC system. The first AP system was designed by Kenneth Batcher and implemented in the Goodyear Aerospace STARAN computer, which was specifically designed to support ATC [12,55,72]. A second generation STARAN called the ASPRO [56], also an AP designed at Goodyear Aerospace, was used extensively by the Navy for Airborne Command and Control Systems for over ten years [67].

The hardware enhancement that is required for an AP is a broadcast/reduction network such as described in [44]. This network supports the execution of the below operations, often called the associative operations, in constant time. A list of the associative operations follow [44,66,67,79]:

- Global MAX and MIN: Finds all instances where a maximal (or minimal) value is stored by a processor at a fixed memory location. The processors whose value is maximal (respectively minimal) are active at the end of this operation and are called *responders*. The remaining processors participating in this activity are called *non-responders*.
- AND and OR: finds the global reduction of Boolean values stored in the same memory location of each active processor.

Table 1
Timings of associative functions.

Associative functions	Timings in C^n (μs)	Timings in assembly (μs)
Max	5.257	3.654
Min	5.257	3.654
AND	7.024	NA
OR	7.364	NA
Associative search	29.2	NA
Any	0.281	NA
Nany	15.147	8.245
Get	13.116	7.876
Next	13.032	7.816
Broadcast	100	NA

- Associative search: finds all instances where a search pattern is matched by the content stored by a processor at a fixed memory location. The active processors whose content matches the search pattern are the *responders* and the processors which did not match the search pattern are the *non-responders*.
- Any-Responder: the ANY operation determines if there is at least one responder after an associative search.
- Pick-One: selects one responder from the set of the responding processors. It is implemented on ClearSpeed using their GET and NEXT operations.
- Broadcast data or instructions from the control unit to PEs.

Because the CSX600 coprocessor is SIMD, only the associative functions need to be emulated. These associative functions have been implemented on the CSX600 efficiently, in both the C^n language introduced in Section 2 and assembly language supported by CSX600 [85–87], mostly by Dr. Kevin Schaffer. To evaluate their running time, we store 30 records in each PE and perform each associative operation once for each of these 30 records. The timing results in microseconds (μs or 10^{-6} s) are shown in Table 1. The NA in the table means that either they cannot be implemented in assembly or they are assembler commands. A command for an associative search is not required as an associative search can easily be made using the C^n language and other associative commands. Although they are not constant time, they are very efficient, and establish that we can efficiently emulate an AP using CSX600. The reason that these functions are not constant time on ClearSpeed is that they are not supported by a broadcast-reduction network, but instead with the swazzle (or ring) network. Clearly, the time to use this network increases linearly as the number of processors increases.

4. AP properties

We discuss some issues regarding AP properties in this section. One is whether the methods used in data transfer of the ClearSpeed emulator is similar to data transfer between PEs and control unit memory in AP. The issue really comes down to whether the AP we plan to use can make the necessary data transfers in sufficient time, as there are no transfer properties assumed for AP. The AP model is a general purpose computational model and does not make any assumptions regarding how the interconnection network is used. Often, the use of the interconnection network in an algorithm can be avoided by using the above associative operations, leaving the running time of this algorithm independent of the network used. Also, the AP does not make any assumptions regarding how data is transferred from the AP to outside buffers or between the control unit memory and the parallel memory. Clearly, these transfer times depend on the data size, and this increases as the application size (e.g., the number of aircraft) increases. While it is very efficient to transfer data between the mono and PE memory in ClearSpeed, we do not know if this rate of transfer would scale if a much larger ClearSpeed system is built. We must point

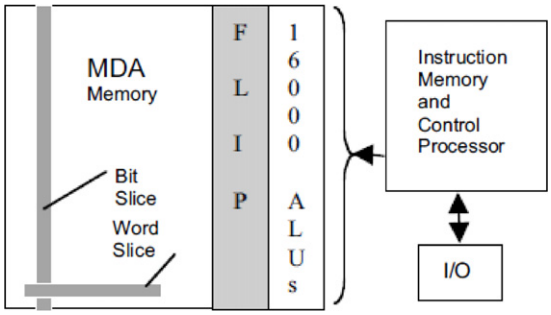

Fig. 3. Flip network of AP.

out the important fact that most and probably all of the non-AP SIMDs that have been proposed and built could not handle the ATC requirement because of I/O limitations. However, the previously built AP machines STARAN [12,55,72] and ASPRO [56] overcame the I/O limitation by the MDA (multidimensional access) memory [12,55,59,66,72] and the flip network (see Fig. 3). The flip network is a corner turning network that provides access to a slice of memory in a SIMD PE for I/O purposes [12,14,66]. The flip network in STARAN and ASPRO was the key to providing high speed I/O movement. The placement of the flip network between processors and their memory allowed very fast movement of data between an outside buffer and the ASPRO/STARAN PE parallel memory. Corner-turned data and the assignment of one record per processor allows multiple PEs to work together to transfer large amounts of data rapidly between PEs and memories. The flip network and MDA allow efficient movement of a record in one PE to an outside buffer. More about this can be found in [66]. Moreover, Jerry Potter describes an alternate design in [66] that allows multiple records to be transferred to an outside buffer. So the transfer of data speed is not a bottleneck for building an associative processor to accommodate realistic size ATC problems. See references [12,11,13–16,59,73,81] to find more information about how to implement AP hardware.

Another issue involving AP properties is how the constant time operations can be supported in AP and how can we be sure they actually run in constant time. The ATC problem is essentially a dynamic database problem. Much of the data in this database changes rapidly, with many parts being updated every one-half second. In order to be able to update and process this data, we need to have some functions that allow us to access, update, change, and add or delete records in this dynamic database very rapidly. In order to support these rapid database operations, we require that these operations execute in constant time. These are the additional functions we require a SIMD to possess in order for it to be called an associative processor. Here "constant time" is defined to be the time that the sequential RAM model requires for a PE to access its memory, the time for AND or OR operations, the time for addition or multiplication of word length numbers, etc. However, there is an additional issue here that does not arise for sequential constant time operations, namely that these functions may involve reductions of vectors with a component value from each processor. Clearly, if the number of processors is allowed to increase without bound, such constant time reductions are impossible. However, if we limit the number of processors to a practical size (e.g., bound by the number of atoms in the observable universe), it is shown in [44] that all of the constant time functions required for an associative processor are possible. However, these functions cannot use a typical interconnection network, as use of these will require non-constant time. Instead these constant time functions can be supported by use of a binary (or n-ary for $n > 2$) broadcast-reduction tree with nodes that have very limited computational capabilities. In fact, this is the

way these constant time operations are supported in both STARAN and ASPRO, which use a 4-nary tree for broadcasts and another 4-nary tree for reductions. The broadcasting/reduction network on the STARAN is constructed using a group of resolver circuits [12,11,13,16]. It is important to keep in mind that all of the above basic operations are implemented in hardware. These features make the AP model a unique parallel computation model that is powerful and feasible. More information about this can be found in [59,57,12,44].

The key to AP hardware design is (1) high memory-to-PE bandwidth, and (2) low level synchronous operation supported by the (i) elimination of branches in low level loops, and (ii) the elimination of low level barrier synchronism. In the STARAN and the ASPRO, these abilities where supported by corner turning the data and assigning one record per processor, the multi-dimensional array memory (and flip network) and mask register hardware. Specifically, the mask register allows additional parallel searching and processing on subsets of the original search with no branching for special cases. After the search phase of an search, process, retrieve (or SPR) cycle is complete, associative SIMDs can either continue data parallel processing or select a single record to process sequentially with essentially no overhead. These attributes allow associative SIMDs to process all the records simultaneously in a file using data parallelism. More information about this and other issues addressed in this section can be found in the book titled *Associative Computing* by Jerry Potter [66].

5. Advantages of AP over MIMD for ATC

Since SIMDs have only one instruction stream (IS) or control unit, control-type communication between processors is completely eliminated. Communication between the IS and processors occurs in constant time using its broadcast/reduction network. Data communication between processors is completely deterministic and a tight upper bound can be calculated for the worst case. As a result, a numerical worst case time required for communications can be accurately calculated. The AP has some important advantages over MIMD including low overhead synchronization, deterministic hardware, much faster communication, predictable worst-case running time, much wider "memory to processor" bandwidth, constant-time broadcasts and reductions, rapid I/O, and elimination of the following: race conditions, data dependencies, shared resource management, sorting, indexing, cache and memory coherency problems, etc.

When solving a problem, MIMD systems often use software to solve additional problems repeatedly which is not needed in a sequential solution of the original problem. Examples of these types of this "additional software" are dynamic scheduling, load balancing, shared resource management, memory and cache coherency management, preemption, synchronization, priority inversion, sorting, indexing, multi-tasking and multi-threading management software, etc. [58]. Several of these types of difficulties were identified in scheduling periodic real-time tasks on MIMD in [49]. A number of these are problems that have been shown to be multiprocessor NP-hard problems (e.g., [36]). In [36], the definition of multiprocessor is a parallel computer that uses message passing or shared memory, so basically it is a MIMD. To avoid confusion in this paper, we will treat multiprocessor and MIMD as interchangeable terms when discussing NP-hard problems, as in [36]. SIMDs do not ordinarily need to use this additional software. Most sequential solutions to a problem can usually be used to create a similar AP solution with roughly the same number of lines of code. AP solutions are often shorter and simpler than the sequential solutions, as the constant-time associative search AP property can be used to eliminate the use of sorting and linked lists to organize and locate items. Also, the additional AP properties often allow simpler and more efficient solutions for problems than with a SIMD that is not an AP.

6. ATC system design

6.1. ATC data flow

The overall system design for an ATC system is shown in Fig. 4. The executive box controls the single instruction stream of AP using static scheduling. All control paths are from ATC in the executive box to all of the tasks, e.g., report correlation and tracking, etc. Controller input simulates sporadic requests, e.g., weather change, controller input, etc. Radar reports data are transferred from the host to the CSX600 PEs. Tracks are simulated from flight plans in the PEs. The radar reports and tracks are used for report correlation and tracking task. The outputs of tracking task are used for cockpit display, controller display update, terrain avoidance, conflict detection and resolution (CD&R) and final optimization. The results of terrain avoidance, CD&R and final approach are used for cockpit display and controller display update. The results of terrain avoidance and CD&R are used for automatic voice advisory that transfers results to automatic voice advisory driver in the host and produces voice output. The resolution advisories of CD&R task are sent to controllers.

6.2. Static scheduling

The eight ATC real-time tasks can be statically scheduled on the CSX600 using the static schedule for the AP discussed in [59,86,87]. The eight tasks and the periods used for each are (1) report correlation and tracking is executed every 0.5 s; (2) cockpit display, (3) controller display update and (4) sporadic requests are executed every second; (5) automatic voice advisory is executed every 4 s; (6) terrain avoidance, (7) conflict detection and resolution, and (8) final approach are executed every 8 s. An 8 s period is split into 16 one-half second periods. Task 1 is executed during each half-second period. Tasks 2 and 3 are executed during the 1st, 3rd, 5th, 7th, 9th, 11th, 13th, and 15th half-second periods. Task 4 is executed during the 2nd, 4th, 6th, 8th, 10th, 12th, 14th and 16th half-second periods. Task 5 is executed in the 4th and 12th half-second periods. Task 6 is executed in the 8th, task 7 is executed in the 14th and task 8 is executed in the 16th half-second period.

6.3. Scaling up from CSX600 to a realistic size AP

We use the present ClearSpeed System to show that our ATC solution is feasible. However, our purpose is to propose building an AP whose size is appropriate for ATC, not a larger CSX600 board or multiple CSX600 boards because some of the "disadvantages" in MIMD systems may show up in a system with multiple CSX600 boards. As described in Section 1, our ideal AP has at least 14,000 PEs with only one aircraft per PE. Moreover, the memory size and speed of the PEs and of the control unit can be chosen to optimize the ability of this large AP to handle the realistic size ATC system. Although we are unable to prove it is possible to build this ideal AP using our CSX600 simulation, STARAN and ASPRO have already met the goal of supporting 14,000 aircraft simultaneously [12,59,57]. A similar processor, the MPP [12,11,13–16], with 16,384 PEs, was delivered in 1982. While the MPP was not an AP as it did not have the hardware reduction network, this feature could easily have been included. Even larger SIMD machines have been built. In particular, Thinking Machines CM-2 was delivered in 1987 and had 64K processors. Paracel developed a parallel processor with one million processors that was generally believed to be a SIMD.

Fig. 4 shows the overall system design and data flow based on AP architecture. Our solutions to ATC tasks for the CSX600 in Section 7 are easy to implement on the AP with 14,000 PEs. Refs. [12,59,57] have timing results of previous AP with 16,000 PEs. For example, Table 3 in [59] shows that the STARAN AP executed

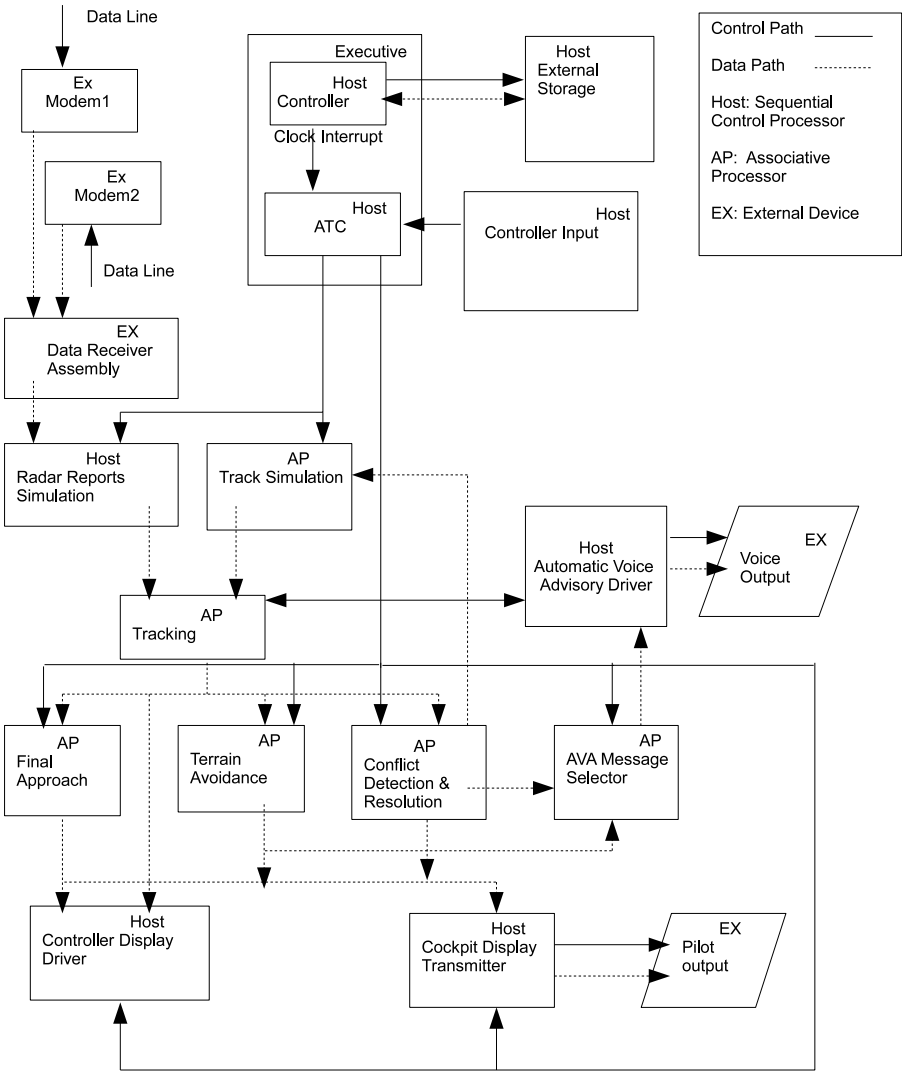

Fig. 4. Overall ATC system design.

a similar set of ATC procedures in 4.52 s, leaving 3.48 unused seconds remaining. That table is simplified to be Table 2 in this paper, which provides worst case execution times for statically scheduled ATC tasks with 14,000 total aircraft, 12,000 sensor reports/s, 6000 controllers and 8 s major period. The data transfer of large records in real-time is not their bottleneck because an AP can bring data in and out efficiently (please see details in Section 4). Additional reasons that the AP model can solve the ATC problem efficiently are its constant time associative operations, SIMD execution style, and extremely wide memory bandwidth. These have eliminated a number of difficulties that have to be handled in MIMD architectures. So an AP with 14,000 PEs using modern technology can be built and can easily meet all of the deadlines, as this has already been done in the past with older technology.

Table 2
Statically scheduled solution time for worst case environment of ATC.

Tasks	Period (s)	Proc time (s)
Report correlation and tracking	0.5	1.44
Cockpit display	1.0	0.72
Controller display update	1.0	0.72
Aperiodic requests (200/s)	1.0	0.4
Automatic voice advisory	4.0	0.36
Terrain avoidance	8.0	0.32
Conflict detection and resolution	8.0	0.36
Final approach (100 runways)	8.0	0.2
Summation of tasks in a period		4.52

7. AP solution for ATC tasks implemented on CSX600

This section describes the algorithms for each of the eight real-time tasks, report correlation and tracking, cockpit display, controller display update, sporadic requests, automatic voice advisory, terrain avoidance, conflict detection and resolution (CD&R) and final approach (runway optimization). The solutions are implemented on the CSX600 architecture, but it is easy to scale up from 96 PEs to an AP with 14,000 PEs using similar algorithms. The best algorithm will always depend on the exact architecture of the AP. For instance, the use of the swazzle network in CSX600 is more efficient here but in a true AP, ordinarily each radar would be broadcast from the host to all PEs simultaneously, as this can be done in constant time on a true AP.

7.1. Report correlation and tracking

The main idea of this task is from [29,59,57]. The report correlation and tracking task is executed every 0.5 s. The input data are radar reports that are simulated in the host and track records in PEs. If total time consumed is considered, this is easily

the ATC task that consumes the most time, as it is performed much more frequently than the other tasks. It is challenging because each report has to be compared with each track, and some aircraft are changing flight mode. We use the SIMD architecture to do the task. First, we create a box for each report and each track and check whether they intersect. It is very efficient because of the features of AP. Second, increase the box sizes of tracks that have not correlated with any reports and then check again because some aircraft might accelerate or perform a maneuver at that time. The details of this task are shown in Algorithm 1. This task can be done both accurately and within deadline because of the SIMD features of the CSX600.

Algorithm 1 Algorithm for Aircraft Tracking

1: Radar reports are transferred from host to mono memory, then distributed from mono to PEs.
2: **for** $i = 1 \rightarrow 96$ in parallel **do**
3: Boxes are created around each radar report and each track in each PE to accommodate report and track uncertainties.
4: Check intersection of each report box with every track box in each PE.
5: If there is an intersection, the radar report and the track are correlated. The *match_count* of this report is incremented, which indicates that it correlates with one track, and its ID and positions are entered into the correlated track's record.
6: All radar reports in each PE are transferred to the next PE using the ring/swazzle network, and steps 3 to 6 are repeated.
7: **end for**
8: After the 96 iterations, all reports have been compared with all tracks. A track that is not produced by noise might not correlate to any reports because the aircraft that it corresponds to is maneuvering.
9: Double the box sizes of tracks that have not correlated with any reports to increase their probability of intersecting a report box and repeat steps of *for* loop above to compare them with uncorrelated reports.
10: Triple the original box sizes of tracks that have not correlated yet, and run the algorithm again.
11: After 3 rounds, if there are still any uncorrelated reports, they are used to start new tracks for potentially new aircraft in this sector.

If a track created for a potentially new aircraft does not correlate with any aircraft after several passes through Algorithm 1, it is viewed as being due to noise and dropped. Note that if this algorithm were to be executed on an AP, it would probably be modified so that each radar report would be broadcast in constant time to all PEs and processed prior to broadcasting the next radar report to the PEs. The modification of the algorithms in this section to run on an AP will be easy but will depend on the exact architecture of the AP.

The altitude and the aircraft ID information is obtained from beacon or secondary radar. Further information about this is included in Section 8.2.7.

7.2. Cockpit display

First the associative operation PickOne is used to select one aircraft. Next, the broadcast operation is used to broadcast the x, y and altitude coordinates of the plane picked in the previous step. For each of its aircraft records, each processor computes the x-distance, y-distance and altitude distance between the location of its aircraft and the location of the aircraft broadcast. Then the processor identifies the aircraft that are approaching this aircraft. This is done by using the conflict detection algorithm covered in Section 7.6.1, find all aircraft that will be within 2×2 nm in x and y and within 1000 ft in altitude in 2 min. Next, transfer these selected aircraft's identity, x, y positions, altitude, velocity, heading, conflict information, etc. to the display server. Finally, use the conflict resolution algorithm in Section 7.6 to obtain a conflict advisory and transfer it to the server. This algorithm uses the SIMD architecture to parallelize the computation and to improve its efficiency.

7.3. Controller display update

This task is similar to cockpit display. We transfer the updated flight identity, positions, altitude, speed, heading, etc. from PEs to the ClearSpeed server, which plays the role of the controller display in this simulation. It uses the SIMD architecture to speed up the display process.

7.4. Automatic voice advisory

Automatic Voice Advisory (AVA) automatically advises an uncontrolled flight (VFR) of near term conditions of other aircraft and terrain by voice. This task is simulated by having the ClearSpeed server print advisories of conflict detection and resolution, terrain avoidance tasks, etc. For example, if there is an aircraft that is approaching the aircraft called, the message might be "aircraft at 4 miles ahead, 4500 ft above, in 1 min"; if the aircraft called is heading for a terrain, the message might be "terrain, 4 miles, 3100 ft ahead". We use an AP style of computation to do this efficiently.

7.5. Sporadic requests

Sporadic requests include information requests or changes in data. For example, aircraft have to avoid an area that has bad weather, aircraft make maneuvers to avoid bad weather, or controllers make a request for runway usage, etc. This task is executed once every second. Although the requests are not processed immediately, they are processed very quickly. We simulate this task as follows. We first process the next unprocessed sporadic task (assuming there are more than one). If it is to divert aircraft so they will miss a storm area, all affected aircraft will be processed using the associative operation PickOne to select one aircraft to redirect at a time.

7.6. Conflict detection and resolution (CD&R)

7.6.1. Conflict detection

This paper considers a conflict to occur when two aircraft are predicted to be within a distance of three nautical miles in x and y and within 1000 ft in altitude. To assure timely evaluation we let the detection cycle be eight seconds, and we determine the possibility of a future conflict between any pairs of aircraft within a five minute "look ahead" period (i.e., 300 s). The input data is from the track records in the PEs. We copy each *track*'s *ID*, 3D position X_t, Y_t and H_t, and velocity V_{xt} and V_{yt} at time t to the following variables, respectively, *ID*, X_c, Y_c, H_c, V_{xc} and V_{yc} in the poly structure *trial* in PEs. Initialize their *time_till* which is the aircraft's earliest collision time with other aircraft to 300.00. The intuitive idea is to compare each trial aircraft with all the other ones, but we use the CSX600 architecture to parallelize the computation. The details of the conflict detection algorithm are shown in Algorithm 2 (see Fig. 5), which is also known as Batcher's algorithm.

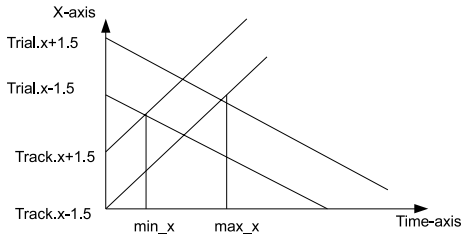

Fig. 5. Conflict detection.

Algorithm 2 Algorithm for Conflict Detection

1: **for** $i = 1 \rightarrow 96$ in parallel **do**
2: **if** for each *trial* and *track* record in each PE, their flight *IDs* are different and altitudes are within 1000 feet **then**
3: Project their positions 5 min ahead, add 1.5 to each x and y coordinate to provide a 3.0 minimum miss distance in each dimension. The x dimension case is shown in Fig. 4.
4: Calculate the *min_x*, *max_x*, *min_y* and *max_y* for minimum and maximum intersection times in x and y dimensions, as shown in equations (1)–(4).
5: Find the largest minimum time *time_min* and smallest maximum time *time_max* across the two dimensions using equations (5) and (6).
6: If *time_min* is less than *time_max*, there is a potential conflict between the aircraft whose ID is *trial.ID* and another aircraft whose ID is *track.ID*.
7: If *time_min* is less than *trial.time_till*, then *trial.time_till* is updated to *time_min*.
8: **end if**
9: All *trial* records in each PE are passed along the swazzle/ring network to the next PE and the above steps are repeated to calculate *trial.time_till*(*trial*, *track*).
10: **end for**
11: After 96 iterations, all *trial* records have been compared with all *track* records. The *time_till* of each *trial* is its soonest collision time with another *track*.

The following formulas are used in Algorithm 2:

$$\min\_x = \frac{|trial.X_c - track.X_t| - 3}{|trial.V_{xc} - track.V_{xt}|} \tag{1}$$

$$\max\_x = \frac{|trial.X_c - track.X_t| + 3}{|trial.V_{xc} - track.V_{xt}|} \tag{2}$$

$$\min\_y = \frac{|trial.Y_c - track.Y_t| - 3}{\left|trial.V_{yc} - track.V_{yt}\right|} \tag{3}$$

$$\max\_y = \frac{|trial.Y_c - track.Y_t| + 3}{\left|trial.V_{yc} - track.V_{yt}\right|} \tag{4}$$

$$time\_\min = \max\{\min\_x, \min\_y\} \tag{5}$$

$$time\_\max = \min\{\max\_x, \max\_y\}. \tag{6}$$

7.6.2. Conflict resolution

We use the SIMD architecture of CSX600 to do conflict resolution accurately and efficiently. Conflict resolution will occur as quickly as possible after conflict detection has been executed, so that each aircraft will have an updated *time_till* value. In practice, conflicts should be reasonably rare. First, we find which aircraft have the shortest conflict time. This is the trial aircraft that will make the initial heading change. Next, the PEs will evaluate in parallel different trial trajectories for the trial aircraft. These trajectories will be created by changing the direction of the current trajectory of the trial aircraft both clockwise and counterclockwise in increments of 5°, up to a maximum increment change of 30°. The various trajectories are rotated through all processors using the ring (or swazzle) network, and the conflict detection algorithm is used to determine the longest time before each trajectory collides with another aircraft. Finally, the maximum of the collision times of all trajectories is determined and used as the best heading change that the trial aircraft can make. If more than one of the best heading change are found, the one with the minimal degree is used. If two have the same degree changes (e.g., $+10°$ and $-10°$ change), one of them is arbitrarily chosen. Although conflicts cannot be fully resolved theoretically, it runs very well in our simulation and during our tests, all conflicts are resolved after several rounds. In an actual implementation, any unresolved conflicts could be resolved by changing the altitude of a plane that still has a conflict after Algorithm 3 is executed.

The algorithm for conflict resolution is described in Algorithm 3. The input data are the *tracks* and *trial* records in PEs. Each PE can have 1–17 *tracks* and *trial* records. Algorithm 3 is based on the CSX600, not an AP. The AP could handle collision avoidance in a much simpler manner, due to its ability to do a constant-time broadcast [59,57,58]. For an AP, the conflict resolution can be included as part of conflict detection. Proceeding one aircraft at a time, each selected aircraft's trajectory information is broadcast (in constant time) to all other aircraft and any potential conflict is immediately corrected as follows. The heading of the initial aircraft is altered by perhaps an increment of 10° and immediately rechecked for a conflict. This procedure continues until a conflict free path with all other aircraft is found with the heading being altered a maximum of 30°. This method is more efficient on an AP since the broadcast of trajectory information of one aircraft's trajectory information can be done in constant time.

7.7. Terrain avoidance

Terrains are lines that make a box shape that encloses a terrain height, e.g., a TV tower is a 1.0 by 1.0 nm box with a height equal to 3100 ft. All terrains and tracks are entered in each PE. Terrain avoidance is as challenging as the report correlation and tracking task. The terrain avoidance algorithm in Algorithm 4 is similar to the conflict detection algorithm. The challenge is the computational intensiveness and we use the CSX600 architecture again to speed up this computation.

7.8. Final approach (runways)

The final approach task is to optimize runway usage. Each flight has a flight plan that specifies its departure terminal, planned departure time, its destination terminal, and planned arrival time. The runways that occur in the region being managed by an ATC system could be distributed among the processors. Each processor manages the information for the runways assigned to it. Here, we assume that there are 96 runways in the sector being managed by this ATC system and assign one runway to each processor. The PE assigned to a runway collects aircraft departure times from this runway and arrival times to this runway and sorts the times. The PEs will instruct the aircraft to increase or decrease their speed to optimize runway usage and also optimize fuel cost. The last step is currently done by controllers manually.

Algorithm 3 Algorithm for Conflict Resolution

1: In parallel, find the minimum *time_till* of all *trial* records sequentially in each PE.
2: Compute the global minimum *time_till* by taking the minimum of the local *time_till* value in each PE.
3: Use the *pick-one* command to select a processor whose local *time_till* is equal to the global *time_till*, and transfer the trajectory record of an aircraft in this PE whose *time_till* is minimal to the mono memory.
4: Create array *projectedpath*[11] in mono memory, copy the best trial aircraft's *ID* and positions to each of the records of *projectedpath*[11], initialize *collision_time*, the soonest collision time with other aircraft to ∞.
5: Each of the *projectedpath*[*i*] represents a path where the best aircraft makes a different heading change from left to right 5°–30° in increments of 5°, alternating between left and right (e.g., 5°, −5°, 10°, −10°, etc.). This will allow numerous different possible paths for the trial aircraft to be evaluated in parallel.
6: Transfer the *projectedpath*[0], . . . , *projectedpath*[11] from mono memory to the PEs in parallel with each PE receiving one *projectedpath*[*i*].
7: **for** $i = 1 \rightarrow 96$ **do**
8: Each *projectedpath* is compared to all aircraft records in each PE with a different *ID* (so that the trial aircraft will not compare to itself) unless their altitudes differ by more than 1000 feet.
9: Calculate minimum and maximum intersection times in *x* and *y* dimensions using the earlier conflict detection equations (1)–(4).
10: Use equations (5) and (6) to get *time_min* and *time_max*. If *time_min* is less than *time_max*, there is a potential conflict between the aircraft whose *ID* is the *trial ID* and this path.
11: Check whether *time_min* is less than the *collision_time* of this *projectedpath*, if so, this *projectedpath.collision_time* is updated to *time_min*. If no conflict is found, the time is not updated and nothing needs to be done.
12: The *projectedpath* records in each PE are then passed to the next PE along the ring network to compare with the records in the neighbor PE.
13: **end for**
14: After 96 iterations, all *projectedpath* records have been compared to all the other aircraft.
15: Find the maximum *projectedpath.collision_time* across all PEs. This path is the best scenario.
16: Change the best trial aircraft's *x* and *y* velocity according to the best scenario path, display the resolution advisory and change the flight plan in the host.

Algorithm 4 Algorithm for Terrain Avoidance

1: **for** $i = 1 \rightarrow 96$ **do**
2: **if** for each *terrain* and *track* record in each PE, the *track* record's height is lower than the *terrain* record's **then**
3: Project the *track* record's position to 2 min in the future, add 1.5 to each *x* and *y* edge of the future position to provide a 3.0 minimum miss distance. The *terrain* records are 1.0 by 1.0 nm boxes.
4: Calculate the minimum and maximum intersection times in both *x* and *y* dimensions.
5: Set *time_min* to be the larger of the two minimum intersection times in both *x* and *y* dimensions in step 4. Similarly, set the record *time_max* to be the larger of the two maximum intersection times in both *x* and *y* dimensions in step 4.
6: **if** *time_min* < *time_max* **then**
7: There is a potential conflict between the *track* and the *terrain*.
8: **end if**
9: **end if**
10: All *track* records in each PE are passed to the next PE and steps 2 to 8 are repeated.
11: **end for**
12: After 96 iterations, all *track* records have been compared with all *terrain* records for terrain avoidance.

7.9. Implementation of algorithm issues

One issue that needs attention is the implementation of these algorithms on real machines. Since our ClearSpeed accelerator has 96 processors on a SIMD chip, the execution of ATC tasks is faster if at most one record of each type is stored on each processor. For example, if there are at most 96 aircraft being tracked at any time, then these can be processed simultaneously in data parallel fashion. However, if there are 100 planes being tracked, then at least 4 processors will have to manage the records for two planes each. For example, when executing the tracking algorithm, all planes with two records will first check their first plane's location against a radar location and then the four PEs managing two aircraft records will check their second plane's location against the radar location. During the time the 4 processors check their second plane's location against the radar location, the other 92 processors without a second aircraft record are inactive. The result is that having even one processor contain two aircraft records will about double the time required to process the aircraft records. More generally, if the maximum number of aircraft records in the processors is k then the time to execute the tracking algorithm will be about k times larger than if there were at most 1 record per processor. As a result, when a new aircraft record is created, it should be added to one of the processors that currently has a minimum number of aircraft records. This can be easily and efficiently accomplished using the MIN AP function.

8. Experimental results

This section describes the results of a set of preliminary experimental results that were conducted to achieve four different goals. First, the experiments provide a proof-of-concept for the proposed ATC system implementation based on the CSX600. Second, the performance and scalability of the proposed approach will be evaluated by performing a comparison between the CSX600-based implementation and the fastest host-only version implementation using OpenMP [62] on a state-of-the-art multiprocessor server system with 8 cores. Similar experiments using OpenMP have been used in image processing algorithms on GPUs [64]. Third, these experiments will provide some initial evidence for the claim that the proposed AP-based ATC system implementation exhibits greater efficiency and a huge increase in the degree of predictability when compared to the MIMD-based solution. Fourth, we show that our model of ATC that consists of eight selected tasks can meet the deadlines for the hard real-time ATC tasks.

8.1. Experimental setup

We are creating a solution for a prototype of ATC since our implementation cannot manage the number of aircraft in a real-world situation. In order to have information about flights that we can control, we simulate the real-world situation by generating aircraft flights in a three dimensional airspace of 1024 by 1024 nautical miles (nm) and 1000–10,000 ft in altitude. The initial positions and realistic velocities of the aircraft are generated randomly and trajectories of aircraft consist of a constant velocity mode and

a coordinated turn mode. A random amount of error is added to the location of each plane to create the radar reports. Since we control this entire process, we can generate different numbers of aircraft to test algorithms and test the limits on the number of aircraft that can be processed fast enough to meet deadlines. Unlike live FAA flight data, when two aircraft are on a collision course, we can alter the flight path of one aircraft to eliminate this problem.

The emulation of the AP solution of ATC uses the ClearSpeed CSX600 accelerator board and is implemented using the C^n language as described in Section 2. The alternative implementation was on an Intel Dual Processor Xeon E5410 Quad Core 2.33 GHz system with 32 GB of main memory and 2 × 6 MB of L2 cache (for each CPU). This system has a total of 8 cores. The implementation was done in C using the gcc compiler version 4.1.3. Both machines' power are similar because their peak performance in doing some embarrassingly parallel computations [82] are almost the same, e.g., Mandelbrot set [27,21]. So our comparison is fair. In this section, MIMD will usually be used to refer to this specific machine. Occasionally, it will refer to a generic MIMD, but these instances should be clear from the context. A single threaded implementation on a single core of the multicore is called *STI*. We denote the multi-threaded implementation on the multicore by *MTI*. We have carefully tuned the OpenMP codes to achieve a good performance, e.g., avoid false sharing, maximize parallel regions, ensure the efficient use of cache, avoid poor load balance problems, etc. Additional information about the techniques used to obtain highly efficient OpenMP programs are given in Chapter 2.5 in [84].

8.2. Experiment results

8.2.1. Comparison of performance on STI, MIMD and CSX600

We scheduled the tasks of report correlation and tracking, terrain avoidance and CD&R on both machines. The report correlation and tracking is executed every 0.5 s, the terrain avoidance every 1 s, and the CD&R task every 2 s. The deadline for each task occurs at the end of each of its periods. None of the tasks can start their executions before their release times and all of them must complete their executions before their deadlines. Each approach was executed for a varying number of aircraft, ranging from 96 to 1824 in increments of 96. For each number of aircraft in the given range, each approach was executed for 100 times. The major period is 2 s and contains one CD&R period, two terrain avoidance periods, and four report correlation and tracking periods. If any of the three tasks misses its deadlines during a major period, we say that the execution has missed its deadline during this period.

We execute the three tasks under all three approaches, *CSX600*, *MTI* and *STI*. The comparisons of maximum execution times are shown in Figs. 6–8. All of these graphs show a significant speedup of the performance of the MTI implementation over the sequential STI implementation so the parallel code for MTI has good speedup. Section 8.2.2 shows the difference between MTI and SIMD more clearly in its three figures.

8.2.2. Comparison of performance on MIMD, CSX600 and STARAN

Timing results for tracking and correlation, CD&R, terrain avoidance and display processing tasks for the STARAN are given in [12,59,57]. It only takes 0.11 s to execute the tracking task for 2000 aircraft for the STARAN. The AP in the following four figures stands for STARAN specifically. Fig. 9 compares timings of the tracking task for the MIMD, CSX600 and STARAN. It only takes 0.1 s to execute the terrain avoidance task for 2000 aircraft in the STARAN. Fig. 10 compares timings of the terrain avoidance task for the MIMD, CSX600 and STARAN. It only takes 0.28 s to

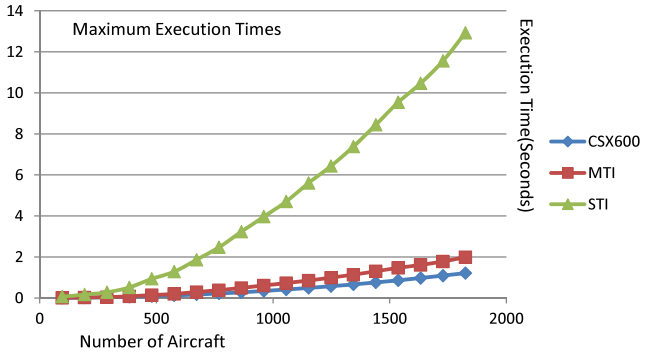

Fig. 6. Timing of tracking task.

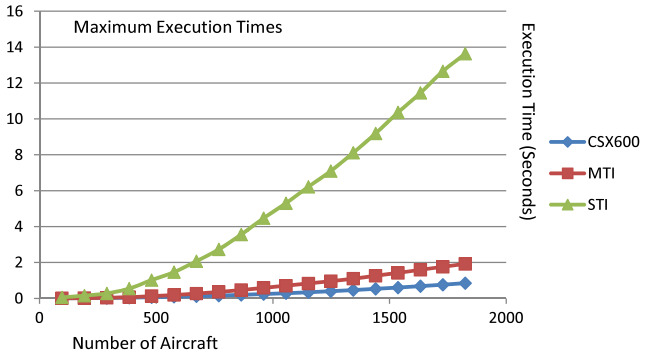

Fig. 7. Timing of terrain avoidance task.

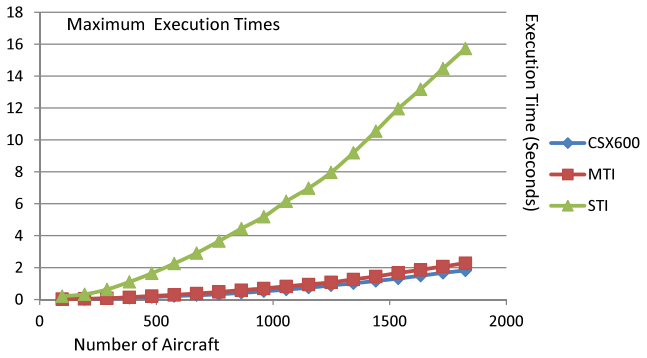

Fig. 8. Timing of CD&R task.

execute the CD&R task for 2000 aircraft in the STARAN. Fig. 11 compares the timings of the CD&R task for the MIMD, CSX600 and STARAN. It only takes 0.16 s to execute the display processing (controller display update) task for 2000 aircraft for the STARAN. Fig. 12 compares timings of this task for the MIMD, CSX600 and STARAN. Although we do not have a modern AP system to test, with today's advanced architecture, we believe that it would execute the ATC tasks more quickly and accurately and at least as predictably, compared to the STARAN.

The comparisons between MTI and CSX600 in Figs. 9–12 are fair, as both are small systems and are doing the entire job. However, the comparisons between the CSX600 and the STARAN are not fair, as the STARAN has only one aircraft per PE, while the CSX600 has an increasing number of aircraft. The CSX600 curves in the three graphs above make it appear that this system is more like a MIMD than an AP, due to the fact that the number of records per processor is increasing. Section 8.2.3 will explain how to compare STARAN with CSX600 fairly and demonstrate how closely our CSX600 emulates the STARAN.

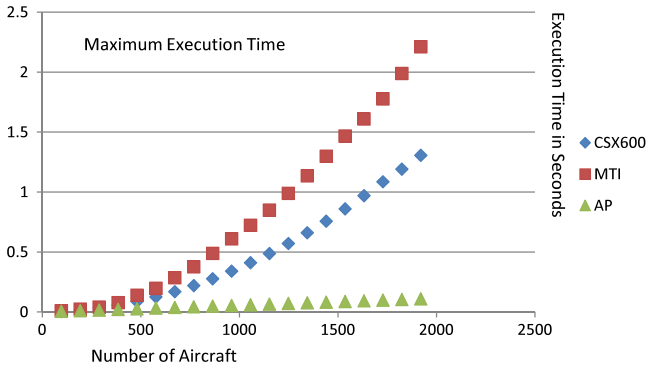

Fig. 9. Comparison of tracking task of MTI, CSX600 and STARAN.

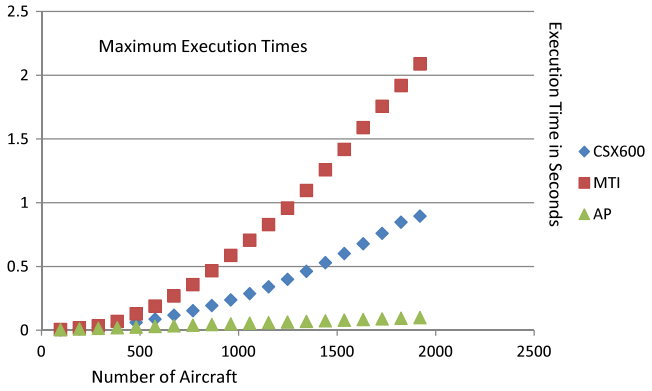

Fig. 10. Comparison of terrain avoidance task of MTI, CSX600 and STARAN.

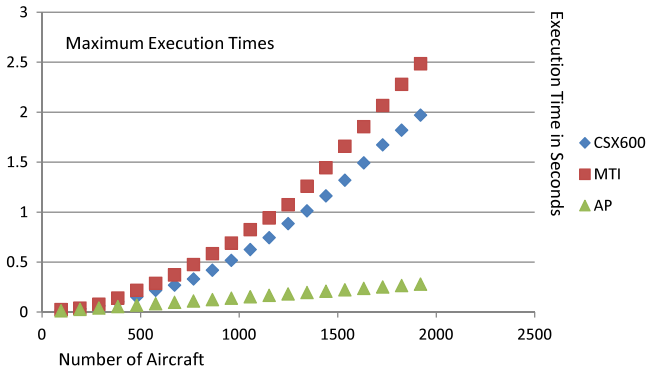

Fig. 11. Comparison of CD&R task of MTI, CSX600 and STARAN.

8.2.3. How close to the AP is our CSX600 emulation of the AP

This section illustrates how closely our CSX600 emulation of the AP performance is to AP performance. Figs. 13 and 14 shows the timings of tracking and CD&R tasks for the total number of aircraft increasing from 10 to 384, i.e., number of aircraft per PE is 1, 2, 3 and 4 in CSX600. We can see that the execution time is basically linear when the maximum number of aircraft per PE does not change, and there is a gap when the maximum number of aircraft per PE increases. Moreover, the slope of each of the line segments is reasonably close to the slope of the STARAN over that interval. These results show that our CSX600 emulation is closely tied to STARAN, and it emulates the solution of STARAN the best when the maximum number of aircraft in each PE is 1.

As explained in Section 8.2.2, the comparison between STARAN and CSX600 is unfair, so we provide a fairer way to compare them. We will use the CSX600 to emulate a Super-CSX600 with enough processors to match the upper bound of the number of planes so that there is at most one aircraft per PE on it. The CSX600 is

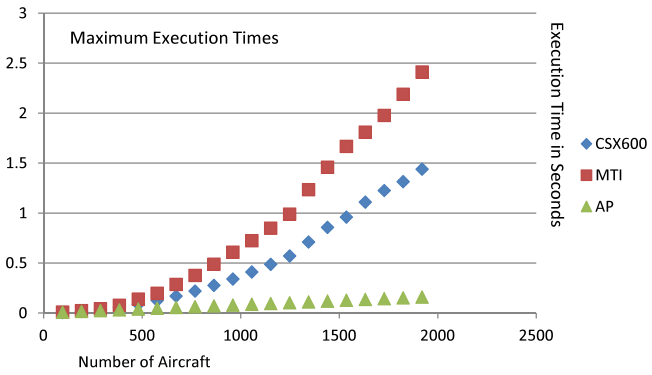

Fig. 12. Comparison of display processing task of MTI, CSX600 and STARAN.

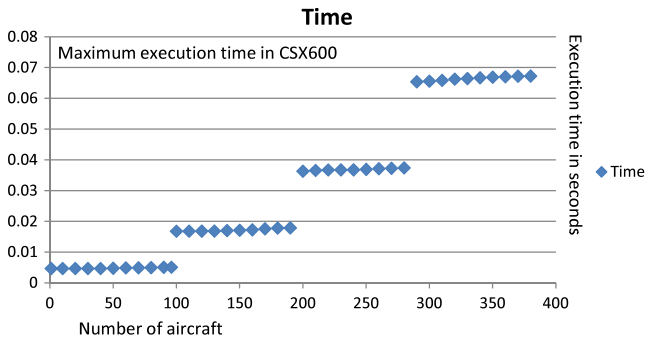

Fig. 13. Time of tracking for number of aircraft from 10 to 384.

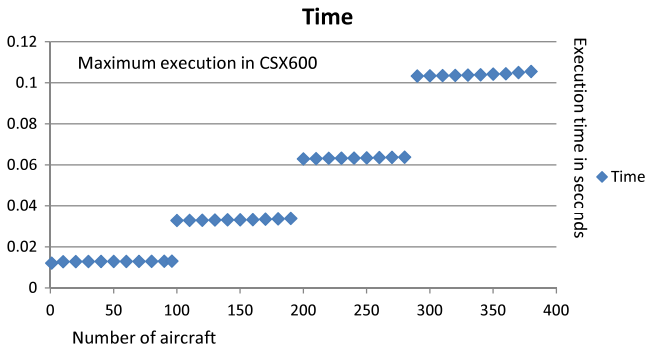

Fig. 14. Time of CD&R for number of aircraft from 10 to 384.

still paying a price for having to do extra moves between mono memory and parallel memory in emulating the Super-CSX600. The CD&R execution time for the Super-CSX600 is obtained by adding the non-parallel computation time to the quotient of the parallel execution time divided by the maximum number of aircraft per PE. The results in Fig. 15 show that the Super-CSX600 is basically linear and reasonably similar to the STARAN, but the cost increases faster for the former due to greater overhead, such as data transfer between mono memory and processors, the non-constant cost of the associative functions, disadvantages of hardware compared to STARAN, etc. There is a significant difference between the AP and the Super-CSX600, but it is dwarfed by the difference between their curves and the MTI curves.

8.2.4. Comparison of predictability on CSX600 and MIMD

The preceding results demonstrate that the performance of CSX600 is better than that of *STI* and *MTI*. We next focus on the predictability of the execution times, which is an important factor in guaranteeing that all the ATC processing can be performed within the specified time bounds. We measure the *Coefficient*

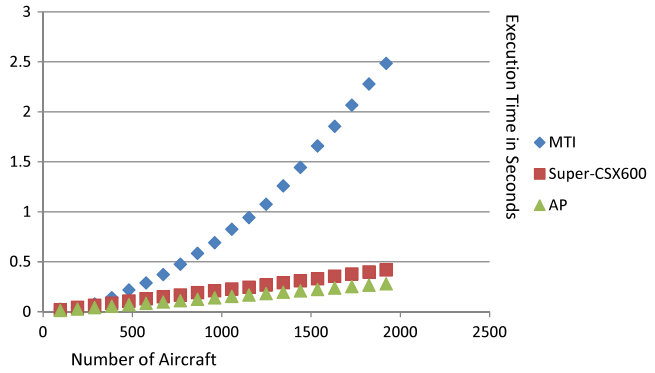

Fig. 15. Comparison of MTI, STARAN and Super-CSX600 with at most one aircraft per PE.

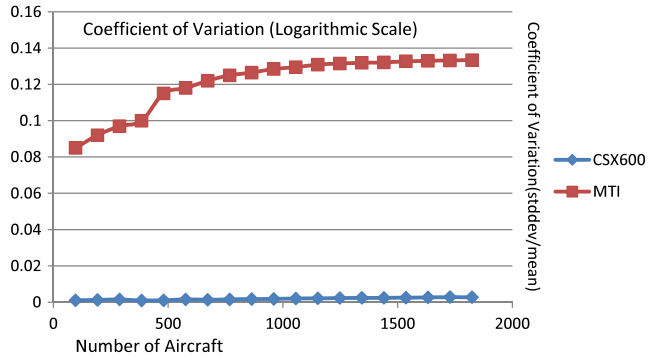

Fig. 16. Predictability of execution times for report correlation and tracking task.

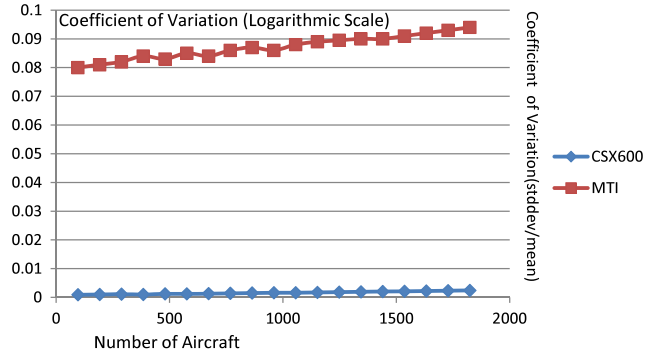

Fig. 17. Predictability of execution times for terrain avoidance task.

of Variation (COV), which is a common normalized measure of dispersion, and is defined as the ratio of the standard deviation to the mean. The smaller the value is, the lower the variance is and so the higher the predictability is. Unlike the standard deviation, the COV is dimensionless, so we think that it is a better tool to use to compare predictability than standard deviation. The COV of the three tasks are shown in Figs. 16–18. Note that the *y*-axis in these figures uses a logarithmic scale. The results clearly show that the COV values for CSX600 are several orders of magnitude below the ones for *MTI*. In addition, the increased variability of MIMD will not be compensated for by the running time as shown in Figs. 9–11, as the MIMD running time increases much more rapidly than CSX600 and AP. Again this is because of the advantages of AP over MIMD mentioned in Section 5. These scenarios are true for not only less than 8000 aircraft, but also for large scales such as 14,000 aircraft.

8.2.5. Comparison of missing deadlines on CSX600 and MIMD

Last, we compare the number of major periods that have missed their deadlines for both CSX600 and *MTI*. The scheduling was

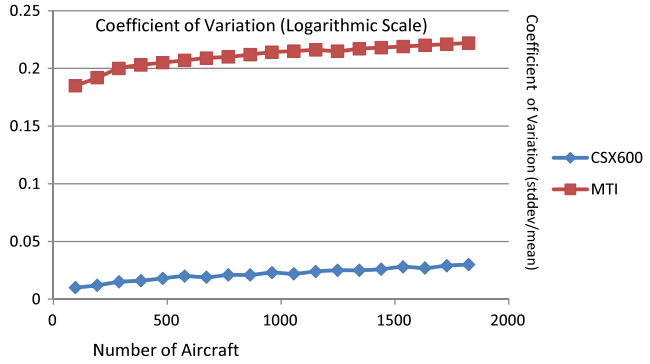

Fig. 18. Predictability of execution times for conflict detection and resolution task.

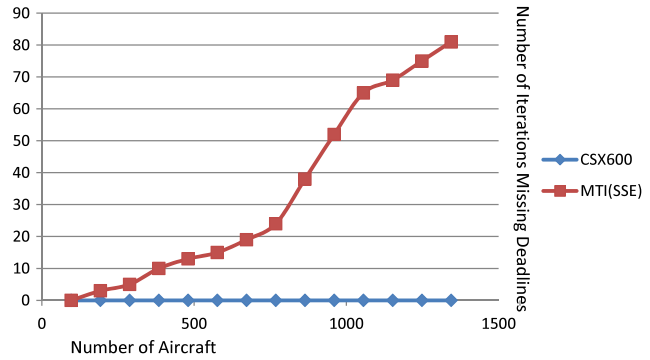

Fig. 19. Number of iterations missing deadlines when scheduling three tasks.

Table 3
Performance of one flight/PE.

Tasks	Exec time (s)	Proc time (s)
Report correlation and tracking	0.00552	0.08832
Cockpit display	0.00272	0.02177
Controller display update	0.00276	0.02209
Sporadic requests	0.00155	0.01244
Automatic voice advisory	0.00544	0.01088
Terrain avoidance	0.00782	0.0782
Conflict detection and resolution	0.01301	0.01301
Final approach (96 runways)	0.00837	0.00837
Total		0.25508

described at the beginning of this section. The results are shown in Fig. 19. When the number of aircraft increases, the number of deadlines missed by the *MTI* execution increases dramatically. However, the CSX600 does not miss any deadline during each major period.

8.2.6. Timings for 8 tasks

In this section we will show that ClearSpeed can perform all the required tasks in our ATC prototype and meet all deadlines for each major period. Table 3 shows the performance of one flight per PE, which is the closest scenario to the AP. The execution time (s) is the time that is spent to execute this task once. The processing time (s) is the total time that is spent for this task in an 8 s period. We can see that all tasks can be done within their deadlines. The total used time is 0.25508 s, which is only 3.19% of the available time of 8 s.

Table 4 lists the worst case timings of eight tasks for 10 aircraft per PE running on the CSX600, which is the maximum number that can be scheduled because each PE only has 6 K of memory. The total processing time is 10.4018 s while the maximum time required by ATC is typically 10 s [59,57,61,34,32]. Since a 10.4 s cycle is only slightly larger than a 10 s cycle, the performance results are relatively good and demonstrate the scalability of this implementation on the CSX600. We can conclude

Table 4
Performance of eight tasks for ten flights/PE.

Tasks	Exec time (s)	Proc time (s)
Report correlation and tracking	0.3409	5.4544
Cockpit display	0.1705	1.364
Controller display update	0.1825	1.46
Sporadic requests	0.0987	0.7896
Automatic voice advisory	0.1586	0.3172
Terrain avoidance	0.2386	0.2386
Conflict detection and resolution	0.5182	0.5182
Final approach (96 runways)	0.2598	0.2598
Total		10.4018

that the CSX600 implementation can meet all deadlines that can be statically scheduled. The ClearSpeed CSX700 provides a larger SIMD system with 192 processors and can double the number of aircraft supported by the CSX600. With a faster 250 MHz clock speed, the CSX700 should either meet or come close to meeting a 10 s deadline for execution of the major period.

The reason that we use 96 in Tables 3 and 4 is that 96 is the maximum we can use without increasing the CSX600 running time of this task by a factor of about 2. Also, this larger number demonstrates the advantage of using a SIMD for a more computationally intensive problem and should be enough to include all of the runways in airfields under the control of an ATC center. The bound of 100 runways is used in both [59,57]. We have also tested executions for 10 runways, which is more practical for an airport size application (as opposed to an entire ATC sector). The result is almost the same since the maximum number of runways per processor is 1 in both cases. This shows that our approach is not only appropriate for large scale applications but also for small scale applications. This is not a limitation of our approach.

8.2.7. Video and actual STARAN demos

In a contract with FAA, Goodyear Aerospace gave a demonstration in 1971 in Knoxville, TN. This unit used a content-addressable memory built with magnetic wire to demonstrate automatic tracking, conflict detection and resolution, terrain avoidance, and automatic voice advisory using only about 4500 instructions. This unit was a predecessor to the STARAN.

As a result of the successful performance of the experimental magnetic SIMD at Knoxville, Goodyear developed a production AP named STARAN. The STARAN system had array modules containing 256 PEs and 1028 bits of storage per PE. The system was programmed by 5 people in 5 months and consisted of 4017 AP assembly instructions, and about 1600 instructions for the control processor [72]. It was delivered for demonstration in 1972 at the International Air Exposition at Dulles Field, and performed as a control center processor. The Edwards AFB radar was used as radar input. All radar data was tracked automatically. Kenneth Batcher provides a detailed account of this demonstration in the "Air Traffic Control" section of [12].

Radar data used in tracking aircraft takes two forms. Both are on a rotating platform (very often the same platform). The first is primary radar which sends out a pulse that is reflected from the target and provides range and azimuth of the target. The second is secondary or beacon radar that sends a message asking the aircraft transponder to reply with one of several possible replies. Usually the beacon requests the aircraft transponder to reply with identity or altitude. Range is also measured along with azimuth. Currently beacon radar is the main signal used in tracking aircraft. Using the altitude data from the beacon radar allow the aircraft to be tracked in three dimensions. Position is provided by the target range and azimuth and the altitude data from the beacon radar allows the aircraft to be tracked in three dimensions.

In the STARAN automatic tracker in 1972, both primary and beacon signals were used for tracking. It is often found that the primary radar gives a more precise measurement — thus a more precise track in 3-D. The STARAN automatic tracker would have tracked all of the 9/11 aircraft after they turned off their beacon. In the future, hackers could generate false beacon signals, which can be detected and ignored if automatic primary radar tracking is used.

NexGen will use GPS positions from each aircraft to provide even better tracking. Additionally, automatic primary tracking can be used with GPS tracking to detect aircraft trying to avoid being tracked or false aircraft signals.

A movie film was taken of the demonstration in 1972. The person appearing in this film is Will Meilander. A data block is connected to each plane. These data blocks are repositioned when needed to avoid having two of them intersecting. A conversion of some parts of this film to DVD, called here the *initial video*, was made in 1987 and is available for viewing at [78] and at the supplementary appendix at the end of the article. The "voice over" narration was added in this video by Will Meilander. An improved version called the *second video* will also be posted at both of these sites as soon as possible. The video shows automatic primary and beacon radar tracking, conflict detection and resolution, and display processing. The demonstration uses 256 flight plans to develop the primary and beacon radar reports. Initially, a ten second major period was used, but later the system clock was first sped up to process major periods in one second and later in one-tenth of a second. The speedup of the system clock by 100 times real time is the equivalent of 25,600 tracks in real time. The real-time, the 10 times speedup, and the 100 times speedup are all captured on the video. What was demonstrated by STARAN in 1972 cannot be done in any ATC system in the world today.

9. Observations and consequences of the preceding results

A recent paper in IEEE Computer [53] observes that the widely accepted RAM model for sequential computation is the reason for the huge progress that has occurred in sequential computation over the past 60 years and observes that a single widely acceptable model for parallel computation could result in similar successes in parallel computation. It describes several features that this parallel model and its target architectures should satisfy, including encouraging parallel solutions to applications that are easy to use, energy efficient, scalable, and highly portable. Moreover, the pitfalls and problems that are identified as desirable to avoid in software production are data dependences, race conditions, load balancing, false sharing, deadlock and non-predictability.

Parallel programming should be as simple and intuitive as sequential programming. Sequential programming is a deterministic and predictable process that arises intuitively from the way programmers solve problems using algorithms. The general programmer should not be required to have an in-depth understanding of the latest developments in hardware design in order to write efficient programs. The current complexity of designing high quality parallel programs makes the process of redesigning and rewriting applications both time-consuming and expensive. The result is that most legacy applications are still waiting to be parallelized. These and related issues are discussed further in [53].

The associative computational model ASC introduced in [67] was designed to support algorithm development for the AP and satisfies all of the above desired characteristics. While the name ASC in [67] originally applied to both APs and multi-APs (which allowed multiple instruction streams), the name ASC was later restricted to the single instruction stream AP and the model for multi-APs was called MASC. Comparisons between the power of these models and other well-known computational models were given in [79]. Possible implementations for MASC are discussed in [22,24,23]. A possible implementation of a multi-threaded associative processor is also investigated in [73].

Next, we consider whether AP can solve a wide range of problems, or whether it is primarily useful in solving only ATC-type problems. A language (also called ASC) with a primer, compiler, and Windows emulator has been designed for ASC [65,66]. Many algorithms have been designed for the ASC model and software projects for the AP platform (often using the ASC language) in order to demonstrate the versatility of the ASC model and the AP system. These include graph algorithms [43,66,67], convex hull [4–7], string matching [30,31], sequence alignment [74,75], NP-complete [50,80], databases [17,18], parallel compilers [1–3], supporting functional and logic-based languages [8,71,76], and supporting expert system languages [39,68,70]. In [12], Kenneth Batcher lists fast Fourier transform, sonar post-processing, string search, file processing, air traffic control, image processing, data management, position locating and reporting, bulk filtering, and radar processing as STARAN applications, and for the first five of these, provides a comparison of the STARAN's performance on this application with several other systems. Additional applications are discussed in [66]. This wide range of AP algorithms and software demonstrate the versatility of the AP to solve a wide range of applications.

The extra problems that MIMDs have to solve as part of their solution to the original problem require a lot of extra work. In addition to the extra problems discussed earlier, the extra work that MIMDs have to do to support the integrity and the ACID properties of a database requires substantial additional work. This requires additional software support that typically involves many of the same types of support software discussed earlier such as synchronization, dynamic scheduling, critical sections of code, sorting, indexing, etc. Also this extra work may involve a substantial increase in communications. All of these problems become much more difficult to handle when the database is dynamic, as records typically have to be added or transferred frequently, and massive updates occur frequently, e.g., every 0.5 s. In contrast, the AP (and SIMDs in general) stores all information about an object such as an aircraft in the local memory of the processor this object is assigned to. This processor executes essentially all of the computations regarding that object, so updates to the database involving this object only require changes to the local memory of that processor.

The graphs in Section 8 of this paper show that the magnitude of this extra work dramatically slows down the time required for MIMDs on ATC-type problems. However, there is nothing highly unusual about the solution to ATC problems, except their hard deadlines. While the heavy real-time nature of the ATC problem and the extremely dynamic database that must be maintained exacerbates the extra work that MIMDs are required to do to solve this problem, this just increases the magnitude of this extra work. It is still the case that for most non-trivial applications, MIMDs have to use a huge amount of extra software and do a huge amount of extra work when compared to SIMDs.

The hardware cost of building a large AP should be much cheaper than building a MIMD that has the capability of solving the same problems due to the simplicity of the AP hardware. While many applications may be reasonable to solve on a MIMD, on important large scale problems that will be expensive and challenging to develop a MIMD solution, it may be considerably cheaper and much more satisfactory to have an appropriate size AP built to handle the problem. The history of the ATC problem provides an excellent justification for considering this alternate approach. The ongoing MIMD ATC solution is widely known for its repeated shortcomings, in spite of the fact that a very large scale project to redevelop this system has been launched about once every ten years for the past 50 years.

10. Conclusions and future work

In this paper, we provided a solution for ATC on the AP and established the feasibility of this solution by showing if this solution is mapped onto the Clearspeed CSX600 emulator of the AP, then this solution works on the AP system. Without many of the previous details of the AP solution, we were able to recreate a large portion of this earlier implementation. This implementation used static scheduling, which is possible due to the deterministic SIMD architecture. This AP ATC software also avoids the inclusion of solutions to the following types of problems (or calls to library functions which solve these problems): load balancing, shared resource management, synchronization, preemption management, sorting, indexing, cache and memory coherency management, false sharing, priority inversion handling, race conditions, data dependencies, deadlocks, etc. In particular, this solution avoids use of solutions or approximate solutions to any of the numerous multiprocessor NP-hard problems of the type given in [36]. The solution used is very similar to a sequential solution for this problem, both in style and in code size. As a result, the size of this software solution is only a very small fraction of the size of the MIMD solutions to the ATC problem. This results in a dramatic drop in the cost in both the cost of creating and in maintaining this software when compared to the MIMD solutions that have been given to the ATC problem.

Further, the AP hardware easily scales to handle larger problem sizes by either increasing the maximum number of records stored in each processor (which will slow down the run-time) or by building a larger AP computer with more processors, which is easy to do. Based on the simplicity of the architecture of the STARAN and ASPRO, which are the only APs that have been built, building a larger AP system should be both easy to do and much cheaper to build than a MIMD system of comparable size. The CM-2 Connection Machine was a SIMD computer built by Thinking Machines in 1987 that had over 64 K processors. Paracel developed a parallel processor with one million processors, which was generally believed to be a SIMD. It would seem reasonable to expect that an AP with a million processors could easily be built currently.

The AP's ability to easily handle the ATC problem would also enable it to easily handle many other real-time problems, both large and small. As a result, this research is relevant not only to the ATC problem, but also to numerous other important applications that involve real-time problems with hard deadlines. For example, command and control systems such as an air defense system would be natural candidates. Other examples may include embedded real-time systems as well.

The ATC problem is a large dynamic database problem. The AP excels in handling the ATC since it was designed to handle real-time dynamic database activities rapidly. For example, locating records with a particular property, reading a value from this record, changing a value in this record, and determining whether a record with a certain property exists are actions that can be done in constant time. By use of the flip network and the multidimensional access memory, large pieces of records can be moved into parallel memory or copied from parallel memory in constant time, making it possible to enter and to ship these records elsewhere rapidly. The AP's capability of handling dynamic databases makes it very useful for numerous other applications.

At this point, a large number of applications that an AP can handle well have been mentioned. The types of applications that APs (and more generally SIMDs) are usually considered to be unable to handle well include multitasking applications that involve multiple tasks that have to be processed concurrently, like ATC. The ATC type of application is normally considered to require a MIMD-type solution. APs cannot execute different tasks simultaneously. Our solution to ATC involves executing all

instances of each task synchronously. For example, the report correlation and tracking algorithm can be executed simultaneously for all aircraft. This paper presents strong evidence that the AP is much better at handling the ATC-types of programs than MIMDs. A key reason for this is that the AP solution does not require the tremendous amount of additional software that is required for the MIMD solution, e.g., dynamic scheduling, load balancing, shared resource management, memory and cache coherency management, preemption, synchronization, priority inversion, sorting, indexing, and multi-tasking and multi-threading software. Also, the AP hardware has important advantages over the MIMD hardware including constant-time broadcasts and reductions, rapid I/O, low overhead synchronization, deterministic hardware, much faster communications, predictable worst case running time, and much wider "memory to processor" bandwidth. Many real-time problems can be handled in a similar manner to the ATC problem, as ATC is very standard model for real-time problems and is discussed in most real-time textbooks. However, the AP can also handle applications which are not real-time but are similar to ATC in that they involve multiple tasks that have to be processed repeatedly. In the AP, each task can be handled by executing all current instances of that task simultaneously.

A natural application area for APs is controlling fleets of unmanned aircraft systems (UASs). In fact, the ATC system developed here with the ClearSpeed CSX600 could be expanded to manage flight control for a small fleet of 96–960 UASs (depending on the number of tasks performed, the size of the records stored, etc.) in applications such as (1) patrolling areas such as international borders and reporting unusual activities; (2) early identification of forest fires, and during actual fires, maintaining updated information about regions that are burning, threatened, etc.; and (3) surveillance of agriculture crops and performing functions like spraying when needed, turning water on at various locations when plants need water, etc. The ClearSpeed CSX700 provides a larger SIMD system with 192 processors and roughly a 20% faster clock time for processors should more than double the capabilities of the CSX600. As ClearSpeed SIMD accelerators are well-known for their very small power consumption, size, and weight, it should be possible to build an AP that also has similar characteristics. For example, it should be possible to build an AP that requires less power than a light bulb and have a sufficiently low weight that would enable it to be easily deployed in the field and still be able to control 1000 UASs in the immediate vicinity. Since UASs will soon be permitted to enter airspace controlled by FAA, many avionics experts expect the total aircraft in the skies to rapidly increase in numbers as the anticipated civilian use of UASs explodes. Since the demands on our ATC system are likely to increase at a much more rapid pace than in the past, the FAA should quickly initiate an investigation into the use of APs for ATC and how to best integrate the APs into the FAA system.

The ClearSpeed has been used extensively as an accelerator to MIMD systems. In several cases, supercomputers increased their ranking in the top 500 by adding multiple ClearSpeed accelerators to their system. MIMD processors could hand off problems that the SIMD accelerator could compute more efficiently and perform other work while waiting for ClearSpeed to return the solution. It may be useful to investigate whether the use of both AP and MIMD hardware in the same system could also prove to be beneficial. Either system could serve as the main system, but would have the option of handing off problems to the other system that it could not handle efficiently. Such a system would need to be able to convert from one mode to the other efficiently or else transfer data from one system to the other efficiently. Perhaps such a combination might be useful in larger systems that were expected to handle a wide range of problems efficiently.

A possibly important extension to our current research would be to consider an implementation of ATC on NVIDIA hardware, which has many SIMD PE groups on its chips. A NVIDIA system with the latest FERMI chip has a lot in common with the MTAP approach of ClearSpeed. Implementing the CSX600 ATC algorithms on this architecture would provide useful information about its ability to provide another useful platform to use for the types of real-time applications mentioned earlier. Another potential project is to investigate implementing our prototype on other parallel systems, e.g., a Convex with FPGA reconfigurable hardware, a Cray System vector processor, IBM's Cell processor, etc. A more complete discussion of conclusions and future work has been described in the author's (i.e., Mike Yuan's) dissertation [84].

Acknowledgments

The authors wish to express their gratitude to ClearSpeed for their gift of the ClearSpeed CSX600 accelerator to use in this research and to Darren Kerybson and then CTO John Gustafson for expediting our obtaining this accelerator. They wish to thank Dr. Kevin Schaffer for allowing us to use the high quality software he developed to perform the associative operations on the ClearSpeed CSX600. They also wish to thank Dr. Frank Drews at Ohio University, Dr. Jerry Potter who is an emeritus faculty at Kent State, and Dr. Oberta Slotterbeck at Hiram College for useful discussions that contributed to this work and to the anonymous referees of this paper for their valuable comments.

Appendix. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.jpdc.2012.05.009.

References

[1] C. Asthagiri, J. Potter, Associative parallel lexing, in: Proceedings of the 6th International Parallel Processing Symposium, IPPS, 1992, pp. 466–469.
[2] C. Asthagiri, J. Potter, Parallel compilation on associative processors, in: Proceedings of the IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques (PACT), North-Holland Publishing Co., The Netherlands, 1994, pp. 315–318.
[3] C. Asthagiri, J. Potter, Parallel context-sensitive compilation, Software-Practice and Experience 24 (9) (1994) 801–822.
[4] M. Atwah, J. Baker, An associative dynamic convex hull algorithm, in: Proc. of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems, Las Vegas, NV, 1998, pp. 250–254.
[5] M. Atwah, J. Baker, An associative static and dynamic convex hull algorithm, in: Proc. of the 16th International Parallel and Distributed Processing Symposium, IEEE Workshop on Massively Parallel Processing, Ft. Lauderdale, FL, Abstract on p. 249, full text on CDROM, 2002.
[6] M. Atwah, J. Baker, S. Akl, An associative implementation of Graham's convex hull algorithm, in: Proc. of the Seventh IASTED International Conference on Parallel and Distributed Computing and Systems, 1995, pp. 273–276.
[7] M. Atwah, J. Baker, S. Akl, An associative implementation of classical convex hull algorithm, in: Proc. of the Eighth IASTED International Conference on Parallel and Distributed Computing and Systems, Chicago, IL, 1996, pp. 435–438.
[8] A. Bansal, J. Potter, Exploiting data parallelism for efficient execution of logic programs with large knowledge bases, in: Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence, 1990, pp. 674–681.
[9] Y. Bar-Shalom, T.E. Fortmann, Tracking and Data Association, Academic Press, 1988.
[10] Y. Bar-Shalom, X.R. Li, Estimation and Tracking: Principles, Techniques and Software, Artech House, Boston, Massachusetts, 1993.
[11] K. Batcher, STARAN/RADCAP hardware architecture, in: Sagamore Computer Conf. on Parallel Processing, 1973, pp. 147–152.
[12] K. Batcher, STARAN parallel processor system hardware, in: National Computer Conference and Exposition, AFIPS74, New York, NY, 1974, pp. 405–410.
[13] K. Batcher, The multi-dimensional access memory in STARAN, in: Sagamore Computer Conf. on Parallel Processing, 1975, pp. 167–168.
[14] K. Batcher, The flip network in STARAN, in: International Conf. on Parallel Processing, 1976, pp. 65–71.
[15] K. Batcher, STARAN series E, in: International Conf. on Parallel Processing, 1977, pp. 140–143.
[16] K. Batcher, The multi-dimensional access memory in STARAN, IEEE Transactions on Computers C-26 (2) (1977) 174–177.
[17] P. Berra, Some problems in associative processor applications to database management, in: Proceedings of the National Computer Conference and Exposition, 1974, pp. 1–5.

[18] P. Berra, E. Oliver, The role of associative array processors in database machine architecture, IEEE Transactions on Computers 4 (1979) 53–61.
[19] H. Blom, Y. Bar-Sharlom, The interacting multiple model algorithm for systems with Markovian switching coefficients, IEEE Transactions on Automatic Control 33 (8) (1988) 780–783.
[20] H.A.P. Blom, R.A. Hogendoorn, B.A. vanDoorn, Design of a multisensor tracking system for advanced air traffic control, in: Y. Bar-Shalom (Ed.), Multitarget-Multisensor Tracking: Application and Advances, Vol. 2, Artech House, 1990, pp. 31–63.
[21] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, J. McDonald, Parallel Programming in OpenMP, first ed., Morgan Kaufmann, 2000.
[22] W. Chantamas, J. Baker, A multiple associative model to support branches in data parallel applications using the manager-worker paradigm, in: Proc. of the 19th International Parallel and Distributed Processing Symposium (WMPP Workshop), 2005, pp. 266–273.
[23] W. Chantamas, J. Baker, A software implementation of a cycle precision simulator of a multiple associative model, in: Proc. of the IASTED PDCS 2010, Marina del Rey, CA, November 2010, (724), 8 pages, Informatics 2010.
[24] W. Chantamas, J. Baker, M. Scherger, An extension of the ASC language compiler to support multiple instruction streams in the MASC model using the manager-worker paradigm, in: Proc. of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2006), 2006, pp. 521–527.
[25] Y.-J. Chiang, J.T. Klosowski, C. Lee, J.S.B. Mitchell, Geometric algorithms for conflict detection/resolution in air traffic management, in: 36th IEEE Conference on Decision and Control, San Diego, CA, 1997, pp. 1835–1840.
[26] Clearspeed technology plc. Clearspeed whitepaper: CSX processor architecture, 2007. URL: http://www.clearspeed.com/docs/resources/.
[27] Clearspeed technology plc. Clearspeed whitepaper: clearspeed software description, 2007. URL: https://support.clearspeed.com/documents/.
[28] Clearspeed technology plc. CSX600 runtime software user guide, 2007. URL: https://support.clearspeed.com/documents/.
[29] E. Eddey, W. Meilander, Application of an associative processor to aircraft tracking, in: Proceedings of the Sagamore Computer Conference on Parallel Processing, Springer-Verlag, 1974, pp. 417–428.
[30] M. Esenwein, String matching algorithms for an associative computer, Master's Thesis, Department of Computer Science, Kent State University, 1995.
[31] M. Esenwein, J. Baker, VLCD string matching for associative computing and multiple broadcast mesh, in: Proc. of the IASTED International Conference on Parallel and Distributed Computing and Systems, 1997, pp. 69–74.
[32] FAA grants for aviation research program solicitation, 2011. URL: http://www.tc.faa.gov/logistics/grants/.
[33] FAA grants for aviation research program solicitation No. FAA-06-01, 2011. URL: http://www.tc.faa.gov/logistics/grants/solicitation/97solict.doc.
[34] FAA's nextgen implementation plan, 2011. URL: http://www.faa.gov/nextgen/media/ng2011_implementation_plan.pdf.
[35] Federal aviation agency 1963 ATC specifications, 1963. URL: http://www.cs.kent.edu/parallel/papers/FAAspecifications.pdf.
[36] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.
[37] J. Gustafson, B. Greer, Clearspeed whitepaper: accelerating the intel math kernel library, Tech. Rep., 2007. URL: http://www.clearspeed.com/docs/resources/ClearSpeedIntelWhitepaperFeb07.pdf.
[38] S. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, P. Dubey, Clearpath: highly parallel collision avoidance for multi-agent simulation, in: ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA), ACM, 2009, pp. 177–187.
[39] T. Hasten, An OPS5 implementation on a SIMD computer, Master's Thesis, Department of Computer Science, Kent State University, 1987.
[40] I. Hwang, H. Balakrishnan, K. Roy, C. Tomlin, Multiple-target tracking and identity management in clutter for air traffic control, in: Proceedings of the AACC American Control Conference, Boston, MA, 2004.
[41] I. Hwang, J. Hwang, C. Tomlin, Flight-mode-based aircraft conflict detection using a residual-mean interacting multiple model algorithm, in: Proceedings of the AIAA Guidance, Navigation, and Control Conference, Austin, Texas, 2003.
[42] I. Hwang, C. Tomlin, Protocol-based conflict resolution for finite information horizon, in: Proceedings of the AACC American Control Conference, Anchorage, Alaska, 2002.
[43] M. Jin, J. Baker, Two graph algorithms on an associative computing model, in: International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA, Las Vegas, 2007, p. 7.
[44] M. Jin, J. Baker, K. Batcher, Timings for associative operations on the MASC model, in: Proc. of the 15th International Parallel and Distributed Processing Symposium, IEEE Workshop on Massively Parallel Processing, San Francisco, CA, 2001, pp. 193–200.
[45] S. Kahne, I. Frolow, Air traffic management: evolution with technology, IEEE Control Systems Magazine 16 (4) (1996) 12–21.
[46] J. Krozel, M. Peters, K. Bilimoria, C. Lee, J. Mitchell, System performance characteristics of centralized and decentralized air traffic separation strategies, in: Fourth USA/Europe Air Traffic Management Research and Development Seminar, 2001.
[47] J. Kuchar, L. Yang, A review of conflict detection and resolution modeling methods, IEEE Transactions on Intelligent Transportation Systems 1 (4) (2000) 179–189.
[48] D. Lainiotis, Partitioning: a unifying framework for adaptive systems I: estimation, Proceedings of the IEEE 64 (1976) 1126–1142.
[49] K. Lakshmanan, S. Kato, R. Rajkumer, Scheduling parallel real-time tasks on multi-core processors, in: Proc. of the 31st IEEE Real-Time Systems Symposium, RTSS'10, San Diego, CA, 2010, pp. 259–268.
[50] J. Lee, Developing parallel SIMD algorithms for the traveling salesman problem, Master's Thesis, Department of Computer Science, Kent State University, 1989.
[51] X. Li, Y. Bar-Shalom, Design of an interacting multiple model algorithm for air traffic control tracking, IEEE Transactions on Control Systems Technology 1 (3) (1993) 186–194.
[52] K. Liu, Composition of Kalman and heuristic tracking algorithms for air traffic control, Master's Thesis, Department of Computer Science, Kent State University, 1999.
[53] A. Marowka, Back to thin-core massively parallel processors, IEEE Computer Journal 44 (12) (2011) 49–54.
[54] E. Mazor, A. Averbuch, Y. Bar-Shalom, J. Dayan, Interacting multiple model methods in tracking: a survey, IEEE Transactions on Aerospace and Electronic Systems 34 (1) (1998) 103–123.
[55] W. Meilander, STARAN an associative approach to multiprocessing, in: Multiprocessor Systems, Infotech State of the Art Reports, Infotech International, 1976, pp. 347–372.
[56] W. Meilander, ASPRO-VME Hardware/Architecture, eR3418-5 LORAL Defense Systems, 1992.
[57] W. Meilander, J. Baker, M. Jin, Predictable real-time scheduling for air traffic control, in: Fifteenth International Conference on Systems Engineering, 2002, pp. 533–539.
[58] W. Meilander, J. Baker, M. Jin, Importance of SIMD computation reconsidered, in: Proc. of the 17th International Parallel and Distributed Processing Symposium, IEEE Workshop on Massively Parallel Processing, Nice, France, 2003.
[59] W. Meilander, M. Jin, J. Baker, Tractable real-time air traffic control automation, in: Proc. of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS, Cambridge, MA, 2002, pp. 483–488.
[60] P. Menon, G. Sweriduk, B. Sridhar, Optimal strategies for free flight air traffic conflict resolution, Journal of Guidance, Control, and Dynamics 22 (2) (1999) 202–211.
[61] M. Nolan, Fundamentals of Air Traffic Control, third ed., Brooks, Cole, Wadsworth, 1998.
[62] OpenMP Website, 2010. URL: http://openmp.org/wp/.
[63] R. Paielli, H. Erzberger, Conflict probability estimation for free flight, Journal of Guidance, Control, and Dynamics 20 (3) (1997) 588–596.
[64] K. Park, N. Singhal, M. Lee, S. Cho, C. Kim, Design and performance evaluation of image processing algorithms on GPUs, IEEE Transactions on Parallel and Distributed Systems 22 (1) (2011) 91–104.
[65] J. Potter, ASC Software, includes a Primer, Windows Compiler, and Windows Emulator, 1992. can be downloaded at: URL: http://www.cs.kent.edu/~parallel/.
[66] J. Potter, Associative Computing: A Programming Paradigm for Massively Parallel Computers, Plenum Press, New York, 1992.
[67] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, C. Asthagiri, ASC: an associative-computing paradigm, Computer 27 (11) (1994) 19–25.
[68] J. Potter, M. Rivett, T. Hasten, Rule-based systems on SIMD computers, in: Proceedings of ROBEXS, 1987, pp. 198–204.
[69] M. Prandini, J. Hu, J. Lygeros, S. Sastry, A probabilistic approach to aircraft conflict detection, IEEE Transactions on Intelligent Transportation Systems 1 (4) (2000) 199–219.
[70] B. Reed, The ASPRO parallel inference engine (P.I.E.): a real time production rule system, Tech. Rep. 85-6048, 1985.
[71] B. Reed, An implementation of LISP on a SIMD parallel processor, in: First annual aerospace applications of AI, Dayton, 1985, pp. 81–90.
[72] J.A. Rudolph, A production implementation of an associative array processor — STARAN, in: The Fall Joint Computer Conference, FJCC, Los Angeles, CA, 1972.
[73] K. Schaffer, R. Walker, A prototype multithreaded associative SIMD processor, in: Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS)-Workshop on Advances in Parallel and Distributed Computing Models, APDCM, Long Beach, CA, 2007, p. 228.
[74] S. Steinfadt, J. Baker, SWAMP: Smith–Waterman using associative massive parallelism, in: IEEE Workshop on Parallel and Distributed Scientific and Engineering Computing, 2008 International Parallel and Distributed Processing Symposium, IPDPS, Miami, FL, 2008.
[75] S. Steinfadt, M. Scherger, J. Baker, A local sequence alignment algorithm using an associative model of parallel computation, in: Proc. of IASTED Computational and Systems Biology, CASB 2006, Dallas, TX, 2006, pp. 38–43.
[76] G. Steele, W. Hillis, Connection machine lisp: fine-grained parallel symbolic processing, in: Proceedings of the 1986 ACM Conference on LISP and Functional Programming, ACM, New York, NY, 1986, pp. 279–297.
[77] D. Sworder, J. Boyd, Estimation Problems in Hybrid Systems, Cambridge University Press, 1999.
[78] The *initial video* of the performance of the STARAN on ATC is posted at URL: http://www.cs.kent.edu/~jbaker/ATC/ and http://youtu.be/IiNwZhdKbVs. This is a 1987 reproduction of part of the original 16 mm film made in 1972. We plan to post an improved version called the *second video* here as soon as possible, 2012.
[79] J. Trahan, M. Jin, W. Chantamas, J. Baker, Relating the power of the multiple associative computing model (MASC) to that of reconfigurable bus-based models, Journal of Parallel and Distributed Computing 70 (2010) 458–466.
[80] D. Ulm, J. Baker, Solving a 2D Knapsack problem on an associative computer augmented with a linear network, in: Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale, CA, 1996, pp. 29–32.

[81] R. Walker, J. Potter, Y. Wang, M. Wu, Implementing associative processing: rethinking earlier architectural decisions, in: Proceedings of the 15th International Parallel and Distributed Processing Symposium, Workshop on Massively Parallel Processing, San Francisco, CA, abstract on page 195, full text on accompanying CDROM, 2001.
[82] B. Wilkinson, M. Allen, Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice Hall, 1999.
[83] L. Yang, J. Kuchar, Prototype conflict alerting logic for free flight, AIAA Journal of Guidance, Control, and Dynamics 20 (4) (1997) 768–773.
[84] M. Yuan, A SIMD approach to large-scale real-time system air traffic control using associative processor and consequences for parallel computing, Ph.D. Thesis, Department of Computer Science, Kent State University, 2012.
[85] M. Yuan, J.W. Baker, F. Drews, W. Meilander, Efficient implementation of air traffic control (ATC) using the clearspeed CSX620 system, in: Proc. of the 21st IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS, Cambridge, MA, 2009, pp. 353–360.
[86] M. Yuan, J.W. Baker, F. Drews, L. Neiman, W. Meilander, An efficient associative processor solution to an air traffic control problem, in: Large Scale Parallel Processing IEEE Workshop at the International Parallel and Distributed Processing Symposium, IPDPS2010, Atlanta, GA, 2010.
[87] M. Yuan, J. Baker, W. Meilander, K. Schaffer, Scalable and efficient associative processor solution to guarantee real-time requirements for air traffic control systems, in: Large Scale Parallel Processing IEEE Workshop at the International Parallel and Distributed Processing Symposium, IPDPS (2012), Shanghai, China, 2012, pp. 1682–1689.

Man (Mike) Yuan is currently a Ph.D. candidate in computer science at Kent State University. He also received his M.S. in computer science from the University of Western Ontario in 2003 and his B.S. in computer science from Hefei University of Technology, China in 2001. His research interests include: high performance computing, parallel algorithms, parallel and distributed computing, associative SIMD computing, and real-time systems, e.g., large scale real-time air traffic control using associative SIMD. He has published papers in IEEE IPDPS and various other international conferences. He is a member of both ACM and IEEE Computer Society.

Johnnie W. Baker received his B.A. degree from Hardin-Simmons University and his M.S. and Ph.D. degree in mathematics from the University of Texas at Austin. He was a faculty member at Florida State University prior to joining the faculty at Kent State University in 1973. Besides computer science, he has published in mathematics, computational chemistry, and bioinformatics. Baker's current research interests in parallel computing include parallel algorithms, parallel modeling and simulation, associative SIMD processors (AP) and multi-AP (or MASC) computing, and real-time air traffic control using an associative processor. He has refereed for many conferences and journals, served as an editor for Parallel Processing Letters for over 15 years, been a continuous member of the organizing committee of both the Massively Parallel Programming and the Large Scale Parallel Processing workshops at the IPDPS Conference, and regularly been a member of several conference and workshop programming committees. He is a member of both ACM and IEEE Computer Society.

Will C. Meilander received his B.S. degree in 1942. He joined the US Naval Research Laboratory and installed techniques for rendering German U-boats relatively useless. He spent 10 years in Civil Service, and joined Goodyear Aerospace in 1952 after studying the "Whirlwind" computer system at MIT. He worked with analog and digital computers and systems and received several patents in the area. He led the Computer Development group in developing STARAN and ASPRO. After retiring in 1983 he became a professor (adjunct) at Kent State University in the Computer Science arena. He taught computer architecture and database management with emphasis on simultaneous parallel processing systems. He retired from Kent State in 2003. His current interest is real-time database design technology offering Associative Processor systems that greatly exceed current real-time multiprocessor capability.